

Practical DevOps for Big Data

Contributors

- [University of Pau](#)— Franck Barbier (Preface)
- [Netfective Technology](#)— Youssef Ridene, Joas Yannick Kinouani, Laurie-Anne Parant
- [Imperial College London](#)— Giuliano Casale, Chen Li, Lulai Zhu, Tatiana Ustinova, Pooyan Jamshidi
- [Politecnico di Milano](#)— Danilo Ardagna, Marcello Bersani, Elisabetta Di Nitto, Eugenio Gianniti, Michele Guerriero, Matteo Rossi, Damian Andrew Tamburri, Safia Kalwar, Francesco Marconi
- [IeAT](#)— Gabriel Iuhasz, Dana Petcu, Ioan Dragan
- [XLAB d.o.o.](#)— Matej Artač, Tadej Borovšak
- [flexiOPS](#)— Craig Sheridan, David McGowran, Grant Olsson
- [ATC SA](#)— Vasilis Papanikolaou, George Giotis
- [Prodevelop](#)— Christophe Joubert, Ismael Torres, Marc Gil
- [Universidad Zaragoza](#)— Simona Bernardi, Abel Gómez, José Merseguer, Diego Pérez, José-Ignacio Requeno

How to Read This Book

This book is about a methodology for constructing big data applications. A methodology exists for the purpose of solving software development problems. It is made of development processes—workflows, ways of doing things—and tools to help concretise them. The ideal and guiding principle of a methodology is to facilitate the job and guarantee the satisfaction of stakeholders involved in a software project—end-users and maintainers included. Our methodology addresses the problem of reusing complex and not easily learned big data technologies to effectively and efficiently build big data systems of good quality. To do so, it takes inspiration from two other successful methodologies: DevOps and model-driven engineering. Regarding prerequisites, we assume the reader has a general understanding of software engineering, and, from a tool point of view, a familiarity with the Unified Modeling Language (UML) and the Eclipse IDE.

The book is composed of eight parts. Part I is an introduction (Chapter 1) followed by a state of the art (Chapter 2). Part II sets our methodology forth (Chapter 3) and reviews some UML diagrams convenient for modelling big data systems (Chapter 4). Part III shows how to adjust UML in order to make it support a stepwise refinement approach, where models become increasingly detailed and precise. Except Chapter 5 introduces the subject, and each of the following chapters (Chapters 6, 7, and 8) is dedicated to one of our three refinement steps. Part IV focuses on model analysis. Indeed, models enable designers to study carefully the system without needing an implementation thereof: a model checker (Chapter 9) may verify whether the system, as it is modelled, satisfies some quality of service requirements; a simulator (Chapter 10) may explore its possible behaviours; and an optimiser (Chapter 11) may find the best one. Part V explains how models serve to automatically install (Chapter 12), configure (Chapter 13), and test (Chapters 14 and 15) the modelled big data technologies. Part VI describes the collection of runtime performance data (Chapter 16) in order to detect anomalies (Chapter 17), violations of quality requirements (Chapter 18), and rethink models accordingly (Chapter 19). Part VII presents three case studies of this methodology (Chapters 20, 21, and 22). Finally, Part VIII concludes the book (Chapter 24) after mentioning future research directions (Chapter 23).

Contents

- I. Introduction
 - 1. [Preface](#)
 - 2. [Introduction](#)
 - 3. [Related Work](#)
- II. DevOps and Big Data Modelling
 - 3. [Methodology](#)

4. Review of UML Diagrams

III. Modelling Abstractions

- 5. Introduction to Modelling
- 6. Platform-Independent Modelling
- 7. Technology-Specific Modelling
- 8. Deployment-Specific Modelling

IV. Formal Quality Analysis

- 9. Quality Verification
- 10. Quality Simulation
- 11. Quality Optimisation

V. From Models to Production

- 12. Delivery
- 13. Configuration Optimisation
- 14. Quality Testing
- 15. Fault Injection

VI. From Production Back to Models

- 16. Monitoring
- 17. Anomaly Detection
- 18. Trace Checking
- 19. Iterative Enhancement

VII. Case Studies

- 20. Fraud Detection
- 21. Maritime Operations
- 22. News and Media

VIII. Conclusion

- 23. Future Challenges
- 24. Closing Remarks

IX. Appendices

- A. Glossary
- B. Index

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data&oldid=3369783

This page was last edited on 30 January 2018, at 18:09.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Preface

Around 2010, the head of the Research & Development unit of Alcatel-Lucent mentioned the expression “Big Data” for a meeting of the ICT scientific council of the French Research Agency (a.k.a. ANR). He told us that networks increasingly convey data on an exponential scale and, as a consequence, ICT scientific issues must be totally rethought and re-addressed in light of the durable “Big Data” phenomenon. Indeed, network players were disturbed in their daily business by IT customers that ask for new means for creating value from data (early, from networks, if possible!). As gold nuggets, data includes “senses”, but immense volumes do not, by definition, facilitate any inherent comprehension, or allow for smart manipulation within reasonable time scales. Worse still, well-known storing, formatting, searching, mining or whatever data technology is no longer suitable for the requirements of “Big Data”!

In charge of defining relevant research directions of funded research programmes for the French public and private sectors, I was first skeptical about the “big” adjective, which confuses me with “raw”, “coarse-grain”, “unstructured”, etc. For example, while everybody uses “Big Data” as a buzzword, few state a more appropriate usage of “Big Data”: are “Big Data” sets, tera, peta, exa, zetta... data sets? In other words, which average data quantity qualifies “Big Data”? Are, for instance, exa (1015) or zetta (1018) “Big Data++” sets or common “Big Data”?

Moving forward, and thus interrogating the High-Performance Computing community that used massively parallel computing infrastructures, my feeling was that this community’s assumption is the fact that all data is available, close, fairly structured... To exaggerate, all data is in high-performance memory at any time! This cloud computing anti-vision disappointed me...

In contrast, interrogating the Massive Data Sets community, they proposed “data methods” in this case, of course, data volumes are “massive” (“huge”? “big”? what else?). Ambiguity arose from the difference/equivalence between “massive” (their chosen name) and “big” in “Big Data”. I quickly understood that their methods cannot be appropriate since they do not approach problems with High-Performance Computing: a proof that their data quantities remain “not so big”. In relation with this loophole, assumptions about “data methods” systematically push SQL, XML, etc. as formatting frameworks, something often far from the “Big Data” reality!

Other disciplines (e-commerce, energy, health, sustainability ...) helped us out of the confusion. They challenge ICT people about the unstoppable digitalization of their own sector, which requires, again, rethinking and re-addressing how systems have to be designed in each sector

Typically, in energy, a smart grid is not an energy distribution system equipped with hardware and software for intelligent monitoring and control. Instead, a smart grid is a “Big Data” processing infrastructure whose intimate relationship with the real world relies on sensors and actuators. The real world deals with energy through a surrounding physical infrastructure. How then, as an innovating system in the energy sector, has a smart grid to be designed? The digital component of the overall system is the core, and, as such, produces constraints on the way the remaining part (surrounding physical infrastructure) has to be composed with! This vision displeases energy engineers, but perfectly illustrates the way “Big Data” problems may be solved: the price of progress!

In fact, “Big Data” as a new scientific discipline is the digitalization of business systems at a never encountered scale. Scientific problems are not computer science problems on one side and/or extant scientific problems in other disciplines on the other side. Solutions depend upon the way scalability issues are properly addressed: scalability is the core of “Big Data” problematics. In other words, data methods or technologies for “lower” scales (i.e., “controllable” data volumes) become obsolete when scales increase. New solutions have to be invented, including software development methods that effectively single out collaborative multidisciplinary behaviors from both the software (devengineers, op. engineers) and sectoral (IT customers, end-users...) sides.

“Practical DevOps for Big Data”, this book, intends to tackle this objective. From proven and relevant software engineering paradigms, namely model driven engineering and DevOps, authors expose results from the European ICT DICE project (<http://www.dice-h2020.eu>). DICE focuses on quality assurance for data-intensive applications. Due to the fact that “Big Data” applications stress a more interdisciplinary approach in tight collaboration with IT customers/end-users, DICE puts forward key criteria like performance and trust. These criteria drives the way software is first designed and next released in production. Criteria

act as early input constraints or quality contracts. As inescapable constraints, DICE wonders how running applications process data and deliver “senses” while satisfying constraints from end to end. The DICE methodology is loopback-based, including monitoring and control for applications in production so that performance anomalies for instance, may be injected in maintenance cycles for quality preservation. Of course, the book also explains how dedicated integrated tools support the methodology for the successful “Big Data” world!

Franck Barbier

University of Pau

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Preface&oldid=3369653

This page was last edited on 30 January 2018, at 11:50.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Introduction

Contents

Big Data Matters

Characteristics of Big Data Systems

Big Data for Small and Medium-Sized Enterprises

References

Big Data Matters

Big data is a major trend in information and communication technologies (ICT). With the constant proliferation of mobiles and sensors around the world, the always growing amount of user-generated contents on the social Web, and the soon advent of the Internet of things, through which our everyday equipments (smartphones, electrical appliances, vehicles, homes, etc.) will communicate and exchange data, this data flood that characterises our modern society opens new opportunities and pose interesting challenges. Governments too, today tend to automate more and more public services. Therefore, the ability to process efficiently massive and ever-increasing volumes of data becomes crucial. Last but not least, big data is a job-creating paradigm shift. In the United Kingdom alone, from 2008 to 2012, the demand for big data architects, big data analysts, big data administrators, big data project managers, and big data designers has respectively risen by 157%, 64%, 186%, 178%, and 329%^[1].

Characteristics of Big Data Systems

An organisation that wants to benefit from big data has to reconsider how its information system is architected. To be big data ready, the system must be designed with three essential key features consistently kept in mind. The first is scalability: infrastructure scalability, data platform scalability, and processing scalability. At the infrastructure level, scalability allows the organisation to be able to add to the system as many storage and computing resources as it wants. Small and medium-sized enterprises (SME) normally have not this power by themselves. But they can use the infrastructure of a cloud provider that offers them the possibility to create more virtual machines or containers at will (in return for payment, of course). At the data platform level, scalability refers to the adoption of data management software inherently distributed, i.e. that disperse data over a network, and that can exploit new available storage spaces resulting from the addition of computers to the network. At the processing level, scalability is about the cooperation of several operating system processes, generally running on different machines, to parallelise a task to be done. An unlimited number of processes can join the effort, particularly when the workload is high.

In sum, the mission of a big data architect is to coordinate the activities of various scalable data platform and processing technologies, deployed on a scalable infrastructure, in order to meet precise business needs. The way the technologies are organised and the relationships between them is what we call an architecture. Some architectural patterns have been discovered and documented, and can be found in the literature. For instance, Nathan Marz's Lambda Architecture^[2]. Although the architecture of the big data system is opaque to end-users, there is one thing they inevitably expect: an overall low latency. So, the second vital property of a big data system is the quality of service (QoS). The QoS is a set of requirements on the values of specific performance metrics like CPU times, input/output wait times, and response times.

The third fundamental characteristic of a big data system is robustness. When some machines or operating system processes of the big data architecture go down, the big data system must continue to function, and the problem must not be noticeable from an external perspective.

Big Data for Small and Medium-Sized Enterprises

We can see from the features described in the previous section that architecting a big data system is out of reach for many SMEs desiring to start competing in the big data market. Firstly, they have to rent the infrastructure of big players. Secondly, it is difficult for them to shortly master big data technologies (sometimes designed by the same big players) due to their high learning curves. Thirdly, it is even more complex to coordinate these technologies and reach an acceptable quality of service.

We believe this situation is not irremediable. To change it, we identified and tried to remove the following hindrances:

1. Lack of tools to reason about possible architectures for a big data system. An architect needs a tool to clarify his ideas about how to organise big data technologies. Modelling languages are perfect for this purpose.
2. Lack of tools able to connect architectural big data models with production. Models are not solely abstract representations; they are based on a concrete reality. In the case of big data, this reality is the production environment, that is, the cloud infrastructure upon which the big data system will be built. An integrated development environment (IDE) should allow architects to move in a user-friendly manner from models (ideas) to production (results). The more a modelling language is formal (i.e. the more its concepts have a precise, even mathematical, meaning) the more it is easy for developers to write programs to exploit, analyse, and transform models, generate software, package it, and deploy it automatically.
3. Lack of tools to refactor deployed big data architecture according to measured runtime performances and pre-planned objectives. When the quality of service obtained is not as good as what models seemed to promise, refactoring tools should be there to correct architectural mistakes.

This book explains our work to overcome these handicaps.

References

1. e-skills UK (2013). "Big Data Analytics. An assessment of demand for labour and skills, 2012-2017" <https://ec.europa.eu/digital-single-market/en/news/big-data-analytics-assessment-demand-labour-and-skills-2012-2017>.
 2. Marz, Nathan; Warren, James (2015). *Big Data. Principles and best practices of scalable realtime data systems* Manning. ISBN 9781617290343
-

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Introduction&oldid=3367084'

This page was last edited on 26 January 2018, at 08:32.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Related Work

Contents

Importance of Big Data for the Business

DICE Value Proposition

Positioning DICE to the market

Business requirements and DICE

Importance of Big Data for the Business

Data are not only part of almost all economic and social activities of our society, but have managed to be viewed as an essential resource for all sectors, organisations, countries and regions. But why this is a reality? It is expected that by 2020 there will be more than 16 zettabytes of useful data (16 Trillion GB). Data are not only part of almost all economic and social activities of our society like velocity, variety and socioeconomic value, flags a paradigm shift towards a data-driven socioeconomic mode which suggests a growth of 236% per year from 2013 to 2020. Thus, data blast is indeed a reality that Europe must both face and endeavour in an organised, forceful, user-centric and goal-oriented approach. It is obvious that data exploitation can be the leading spear of innovation, drive cutting-edge technologies, increase competitiveness and create social and financial impact.

DICE Value Proposition

Value proposition of a business initiative is the most critical factor that must be defined early in the project. Value proposition should be a brief but at the same time comprehensive statement of a project, addressing questions like: what value is delivered to the stakeholders? Which one of the business problems/requirements are solved and which needs are satisfied? What bundles of products and services are offered to each stakeholder segment?

In order to reach to a strong and to the point value proposition that will refer to DICE as a whole bringing to the surface all its innovations, project's tangible assets are produced during its lifetime are identified. Even though it is strongly suggested to continuously monitor the market and revise these assets according to the achievements of exploitation activities during project's lifecycle, the initial analysis should consider project's core objectives and envisioned results. Such kind of aspects are the innovation, the motivation and the added value offered to DICE stakeholders.

Finally, the analysis mentioned before should facilitate the definition of other important parts of DICE exploitation strategy, namely project's market segments, trends and target groups. A qualitative comparison of the features of the existing developments with the expected ones of the DICE ecosystem with respect to stakeholders needs is the key to project long-term sustainability

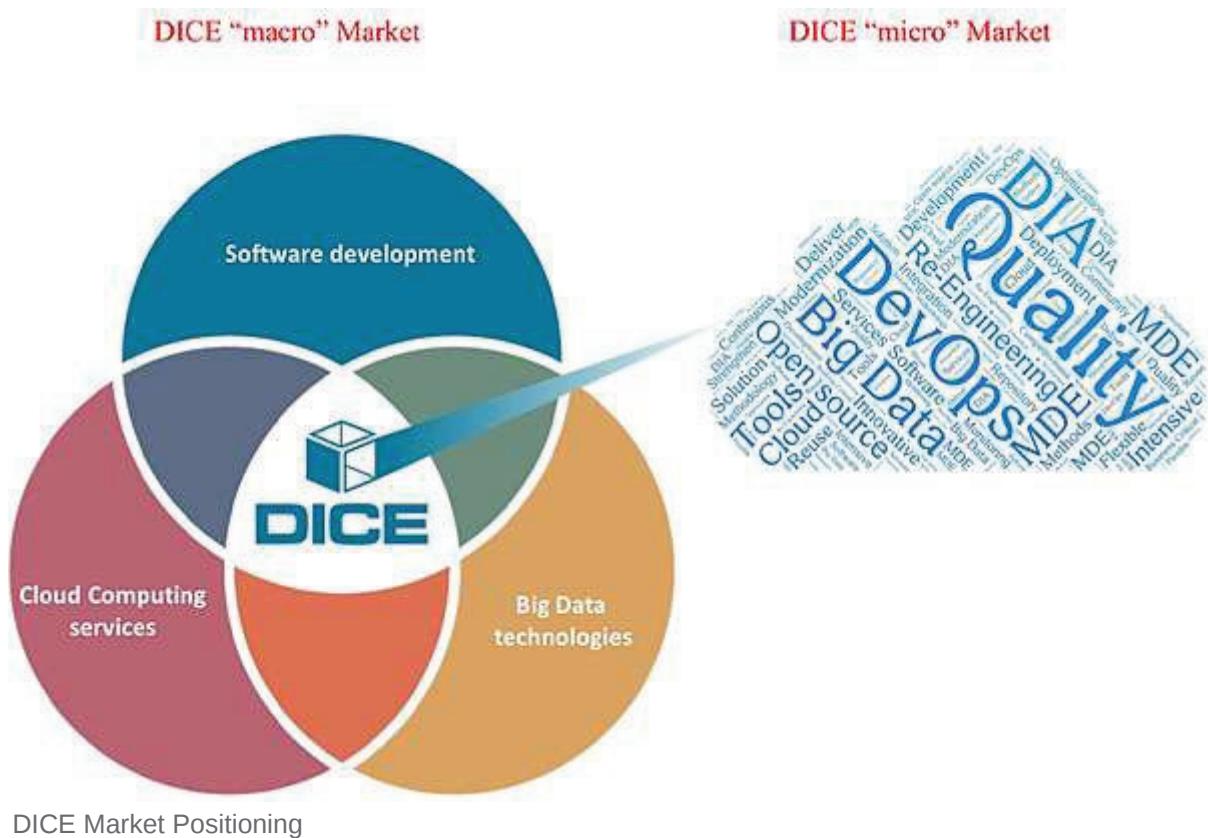
In order to define DICE Value Proposition, DICE exploitation team allocated several tasks to all partners of the consortium. Thus, partners worked on filling in several templates related to the tools they have implementing in the context of DICE, the technologies they have bringing in, the services they have offering, similar initiatives and advantages over them, etc. In the following chapters, we present in details these contributions.

DICE Value Proposition is the following: DICE delivers innovative development methods and tools to strengthen the competitiveness of small and medium Independent Software Vendors in the market of business-critical data-intensive applications (DIA). Leveraging the DevOps paradigm and the innovative Big Data technologies of nowadays, DICE offers a complete open source solution for the Quality-Driven Development of cloud-based DIA.

Positioning DICE to the market

The ultimate goal of analysing the information is to position DICE to the market. Which are the market domains that DICE innovative ecosystem make sense to penetrate, which are the trends of nowadays, is DICE positioned in them, etc. All these question are important to be answered at the early phase of the project so as to steer the development phase in a direction that something innovative is created that will ensure its long-term sustainability. It is obvious that only promising ideas ensure projects viability even after the end of the funding of the project.

From a bird's eye view, the following figure positions DICE to the market by presenting it's the "macro" and "micro" market domains



Business requirements and DICE

DICE can be seen as part of the DevOps movement as it provides a set of tools that facilitate flow of information from Dev to Ops mediated by a model-driven approach and enables operations monitoring and anomaly detection capabilities to facilitate the flow of information from Ops to Dev. Moreover, DICE offers the following tenets in the context of DevOps:

- Thanks to DICE profile, it enables the DIA design with proper operational annotations that facilitate design time simulation and optimizations. ‘Dev’ and ‘Ops’ can work side by side to improve and enhance application design using simulation and optimization tools that offer refactoring of architectural designs.
 - Includes monitoring and tools that exploit the monitoring data to detect anomalies and to optimize the configurations of applications, also the monitoring data is exploited by tools to offer feedback to the architectural design to identify the bottlenecks and refactor architecture.
 - DICE also offers tools for automating delivery and continuous integrations facilitate integrating design-time and runtime tools in DICE and also offering DevOps automations.

Organizations are now facing a challenge to exploit the data in a more intelligent fashion using data intensive applications that can quickly analyse huge volume of data in a short amount of time. However, Big Data systems are complex systems involving many different frameworks which are entangled together to process the data. How will the software vendors perform at increasingly high levels at the exploding complexity? How will they accelerate time to market without reducing quality of the service? This increases Big Data application design and operational requirements, and also demands for a common approach in which development and operation teams are able to react in real time throughout the application lifecycle and assure quality and performance needs.

Therefore, Big Data system development introduces a number of technical challenges and requirements for systems developers and operators that will severely impact on sustainability of Business Models, especially for software vendors and SMEs that have limited resources and do not have the strength to influence the market. In particular the DICE team identified the following problems in practice:

- **Lack of automated tools for quality-aware development** and continuous delivery of products that has been verified in terms of quality assurance. In other words, there exists many different frameworks that software vendors can adopt in order to develop their data intensive applications. However there is no automated mechanism to enable them to verify software quality across such different technological stack in an automated fashion.
- **Lack of automated tools for quality-driven architecture improvements** In order to develop a data intensive application, software vendors require to adopt an architectural style in order to integrate different components of the system. During development, the complexity of such data architecture will be increased and there is no automated tool to enable designer to improve the architecture based on performance bottlenecks and reliability issues that have been detected based on monitoring of the system on real platforms.
- **Lack of tools and methods interoperable with heterogeneous process maturity in industry** Developing and maintaining data-intensive applications is key to IT market diversity spanning from big industrial players and small/medium-enterprises. However there exists a huge variety of software processes in this diversity. Such diversity is seldom taken into account from a methodological and technological standpoint. For example, very few evaluations show the applicability of certain data-intensive design methods within IT companies consistent with Capability Maturity Model Integration (CMMI) Level 4 (Quantitatively Managed) or Level 2 (Managed), yet, the industrial adoption of those design methods may depend greatly on such evaluations. DICE proposes a lightweight model driven method for designing and deploying data intensive applications where models become assets that go hand in hand with intensive coding or prototyping procedures already established in industry (e.g., in CMMI Level 5 players). DICE models play the role of assisting developers in obtaining the desired QoS properties in parallel with preparing their Data-Intensive business logic. This methodological approach offers concrete integration with any CMMI level, from young and upstarting SMEs with initial maturity - these may use DICE at its simplest form, e.g., working with Agile Methods - to large corporate IT enterprises that may require fine-grained and framework specific support to realise quality

Software vendors have always found the way to adapt to new conditions to help organizations adopt and reach their goals. During the last years, the DevOps practices have gained significant momentum. Organizations from across industries have recognized the need to close the gap between development and operations if they want to remain innovative and responsive to today's growing business demands. This new paradigm focuses on a new way of doing things, an approach that tries to improve the collaboration and communication of developers and operators while automating the process of software delivery and infrastructure changes. Therefore, a new ecosystem is needed to facilitate such automated processes by offering the following capabilities:

- **Business Needs and Technical Quality of Service:** All types of organizations need to move fast, and need to align IT assets to their needs (create new assets or adapt existing ones). Therefore, ICT departments need to assure that their ICT platforms and infrastructures are flexible enough to adopt business changes, regardless of the type of IT infrastructures used to create those data intensive applications, whether they are built using open source software or data services offered by public cloud providers. In other words, they should avoid technological lock-in as much as possible.
- **Architecture Refactoring** Efficiency, reliability and safety of applications should be monitored in testing and production environments, with metrics and data that feedback directly and quickly to development teams for faster testing, improvement and adjustment to meet service level goals.
- **Application Monitoring and Anomaly Detection** Access to live data gathered by monitoring engines should also provide performance and reliability data for observed applications in production to gauge the need for identifying outliers and detecting any anomaly that may harm continuous business processes that are dependent to such data intensive applications. Moreover such monitoring data should assist application architects and developers in building toward an optimized target infrastructure.
- **Efficiency and optimal configuration** Big Data applications typically spend expensive resources that are offered to companies via public clouds. Therefore, it is of utmost importance to optimize such applications in order to demand less resources and produce more output. Therefore, organizations require to have automated tools to optimally configure such application without the need to hire experts to optimize their Big Data applications.

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Related_Work&oldid=3365180'

This page was last edited on 23 January 2018, at 11:33.

Text is available under the [Creative Commons Attribution-ShareAlike License](#), additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Methodology

In this chapter, we are going to introduce a way of designing big data applications. The underlying idea is to incorporate techniques from model-driven engineering into a DevOps development life cycle. Why is such an approach suitable and fruitful for data-intensive software? The question is fair, and we shall answer it first. We will start by advocating the use of models in DevOps. We will then look at some benefits of applying model-driven DevOps to big data application construction. Finally we will introduce our methodology proposal and how the DICE IDE gives support to it.

Contents

Model-Driven DevOps

Model-Driven DevOps for Big Data

Methodology Proposal

Architecture Modelling

Architecture Analysis

Architecture Experimentation

The DICE Methodology in the IDE

References

Model-Driven DevOps

In a typical organisation, developers build and test software in an isolated, provisional, development environment—by using a so-called integrated development environment (IDE) such as Eclipse or Visual Studio—, while operators are in charge of the targeted, final, production environment. The latter conceptually comprises all entities with which the application is planned to interact: operating systems, servers, software agents, persons, and so forth. Operators are responsible for, among other things, preparing the production environment, controlling it, and monitoring its behaviour, especially once the application is deployed into it.

Before DevOps, developers and operators usually formed two separate teams. As a consequence, the former were not fully acquainted with the fact that their development environment differed from the production environment. And problems they did not see during the build and test of the application in the development environment suddenly emerged once operators installed it into the production environment. DevOps recommends placing developers—the “Dev” prefix of “DevOps”—and operators—the “Ops” suffix of “DevOps”—in the same team. This way, there is now a continuous collaboration between the two that greatly increases the confidence that every software component created is really ready for production. Indeed, since operators are the ones who understand the production environment very well, their active and constant support to developers results in a program that is designed with a production-aware mindset. The continuity of the cooperation manifests its full virtue when the build, unit testing, integration, integration testing, delivery, system testing, deployment, and acceptance testing phases are all completely automated.



DevOps: from integration to deployment

In DevOps, software components built by distinct DevOps teams are tested in isolation (unit testing). They are then assembled to create a single software (integration). Afterwards, the software is tested (integration testing) and put into a form—often called a *package*—that facilitates its distribution (delivery). Next, the package is tested (system testing) and installed into the production environment (deployment) to be tested one last time (acceptance testing). DevOps prescribes that operators should assist developers in order to automate entirely these stages. When done, continuous integration, continuous delivery, continuous deployment, and continuous testing are respectively achieved.

Model-driven practices have proven their usefulness for designing programs and generating their source codes. Thus, model-driven engineering already covers quite well the build and unit testing of software components. Our idea is to use models to automate integration, delivery, and deployment as well. The DICE Consortium has invented a modelling language for that purpose. (The language does not handle integration though.) Tools have been developed to let developers and operators concertedly model the application, together with its production environment, at different levels of abstraction. These tools interpret models to automatically prepare and configure the production environment, and also to deploy the application thereinto. Here are some advantages of this approach:

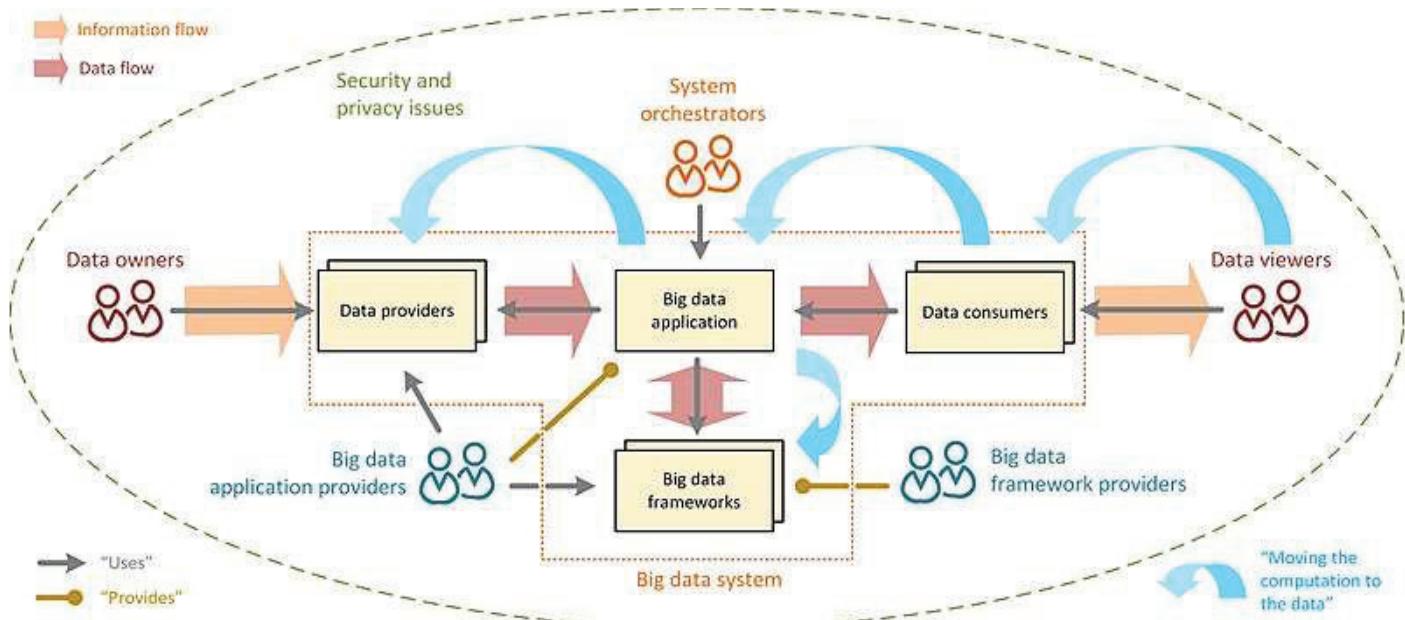
1. Models not only specify the application but also the production environment fit for it.
2. Even more, since properties of the production environment are described, one can sometimes exploit models to simulate or verify formally (i.e. abstractly and mathematically) the behaviour of the application in its production environment before any deployment.
3. A common modelling language for developers and operators contributes to better communications between them. Ambiguities and misunderstandings are diminished because concepts employed in models are clearly defined.
4. It is possible to refactor the application and its production environment directly through their models.
5. The time to market of the application is reduced because some development and operation tasks are now performed by computers.

The DICE Consortium has experimented with these five points and has found the method pertinent. Big data applications generally rely on big data technologies mostly reputed to be difficult to learn. From our point of view model-driven DevOps may help significantly

Model-Driven DevOps for Big Data

The appellation *big data* is used to name the inability of traditional data systems to efficiently handle new datasets, which are massive (volume), arrive rapidly (velocity) from sundry sources and in diverse formats (variety), and whose flow rate may change abruptly (variability)^[1]. Volume, velocity, variety, and variability are known as the four Vs of big data. The orthodox way to deal with them is to scale vertically, that is, to speed up algorithms or hardware (e.g. processors or disks). Such an approach is patently limited and fails without surprise. Big data is a shift towards horizontally scalable parallel processing systems whose capabilities grow simply by adding more machines to the existing collection^[2].

The National Institute of Standards and Technology (NIST) published a valuable big data taxonomy^[3]. A slightly revised version is depicted in the figure below. From top to bottom is pictured the service provision chain: *system orchestrators* expect some functionalities from the big data application, which is implemented by *big data application providers*, with the help of *big data frameworks* designed by *big data framework providers*. From left to right is shown the information flow chain: pieces of information are presented by *data owners* to *data providers*, which digitize them and transfer them to a big data application for processing. Its outputs are got by *data consumers* which may convey them to *data viewers* in a user-friendly manner. The different activities of these nine functional roles are encompassed by security and privacy issues.



Revised NIST big data taxonomy

There is always a social aspect to big data. System orchestrators, big data application providers, big data framework providers, data owners, and data viewers are normally people or organisations. That is why a dedicated symbol represents them. Data ownership is particularly a judicial notion not easy to define comprehensively. A basic definition could be:

Data owner

A *data owner* is a person or an organisation to which a piece of data belongs and which exercises its rights over it. It has legal power to prevent someone else to make the most of its data, and it can take court action to do so—customarily under certain conditions.

People, scientists, researchers, enterprises, and public agencies obviously are data owners^[3]. Data ownership is shareable. Indeed, before using a big data system—e.g. Facebook—a data owner commonly must accept an agreement by which s/he and system orchestrators will be bound. Sometimes, using the system implicitly implies acceptance of the agreement. To some extent, this agreement may concede data ownership to system orchestrators. For example, every internaut who has filled online forms asking personal data probably has seen on one occasion an “I agree to terms and conditions” checkbox. Frequently, these terms and conditions ask the user permission to do specific things with his data. By clicking the checkbox, the user may, for instance, grant system orchestrators the liberty to disclose it to business partners. The role of system orchestrator can be the following:

System orchestrator

A *system orchestrator* is a person or an organisation whose position allows it to participate in the decision-making process regarding requirements of a big data system, as well as in monitoring or auditing activities to ensure that the system built or being built complies with those requirements^[3].

System orchestrators settle, for example, data policies, data import requirements, data export requirements, software requirements, hardware requirements, and scalability requirements. We find among system orchestrators: business owners, business leaders, business stakeholders, clients, project managers, consultants, requirements engineers, software architects, security architects, privacy architects, and network architects^[3].

Data viewer

A *data viewer* is a person or an organisation to which the big data system communicates some information.

In the figure, the boxes denote hardware or software. Data providers, data consumers, big data applications, and big data frameworks are all digital entities. Their job is to cope with data. When we say *data*, we always mean digital data. We use the term *information* to designate non-digital data. And we call *knowledge* every piece of information that is true. *V*racity—yet another V—is an important concern in big data.

Data provider

A *data provider* is a hardware or software that makes data available to itself or to others^[3]. It takes care of rights of access, and determines who can read or modify its data, by which means, and what are allowed and forbidden exploitations.

Database management systems match this definition. Data providers have many methods at their disposal to transmit data: event subscription and event notification, application programming interface, file compression, Internet download, and so on. They can equally decide to create a query language or another mechanism to let users import processing without fetching data. This practice is described as moving the computation to the data rather than the data to the computation^[3], and is represented in the above depicted figure by directed arcs. A data provider may turn information entered or taken from data owners into digital data that can be processed by computers. In that case, it is a boundary between the big data system and the non-digital world. This is, for instance, the function of a dialog box or an online form. All capture devices such as video cameras and sensors are data providers too.

Big data application and big data application provider

A *big data application* is a software that derives new data from data supplied by data providers. It is designed and developed by *big data application providers*.

A big data application is a special kind of data provider—namely a data provider that depends on other data providers. But not all data providers are big data applications. For example, a database management system is not a big data application because it does not generate new data by itself. Big data applications usually perform the following tasks: data collection from data providers, data preparation, and analytics. Data preparation occurs before and after the analytical process. Indeed, as the saying goes, garbage in, garbage out. Likewise, bad data, bad analytics. Hence the necessity to prepare data. For example, data coming from untrustworthy data providers and incorrectly formatted data should be discarded. After the analysis, a big data application may arrange the results so that a data consumer will display them more easily and meaningfully on a screen. All these tasks exist in traditional data processing systems. Nonetheless, the volume, velocity, variety, and variability dimensions of big data radically change their implementation^[4]. In DevOps, developers and operators are big data application providers.

Big data framework and big data framework provider

A *big data framework* is an infrastructural or technological resource or service that imparts a big data application with the efficiency and horizontal scalability required to handle ever-growing big data collections. It is made available by *big data framework providers*.

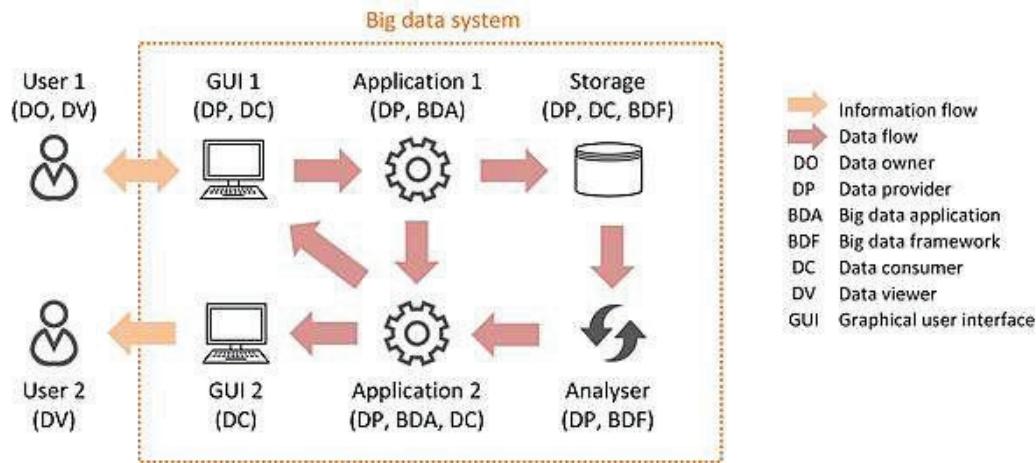
The NIST classified big data frameworks as infrastructure frameworks, data platform frameworks, or processing frameworks^[3]. An infrastructure framework provider furnishes a big data system with the pieces of hardware it needs: storage devices, computers, networking equipments, cooling facilities, etc. The infrastructure may be hidden—clouded—behind a service delivered through an application programming interface. This service may, for example, allow users to remotely create, start, stop, and remove virtual machines or containers on demand. Data centres and cloud providers belong to this category of big data framework providers. Data platform frameworks are data storage programs that capitalise on a network to spread data over several machines in a scalable way. Thus, the more machines we connect together, the more storage space we get. The Hadoop distributed file system

(HDFS) and Apache Cassandra are both data platform frameworks. Processing frameworks, like Hadoop MapReduce and Apache Spark, distribute processing in place of data. The addition of machines must not be detrimental to the efficiency of data retrieval and processing. Instead, a better performance should be observed. Open source communities innovate a lot to improve data platform and processing frameworks.

Data consumer

A *data consumer* is a software or hardware that uses or receives outputs originating from a big data application.

The nine functional roles are not mutual exclusive. It is theoretically possible to be a system orchestrator, a data owner, a data viewer, a big data application provider, and a big data framework provider at the same time. And a program can simultaneously be a big data application, a data provider, a data consumer, and a big data framework. As an example, let us consider the fictitious scenario shown in the following figure:



Example of a NIST big data system

Two users *User 1* and *User 2* are connected to a big data system. They both interact with it by means of a graphical user interface. Let us imagine *GUI 1* is a website browsed by *User 1*, and *GUI 2* is a desktop application run by *User 2*. *Application 1* and *Application 2* are two big data applications. *Storage* and *Analyser* are respectively a data platform framework and a processing framework. Information entered by *User 1* is transferred to *Application 1* by *GUI 1*. *Application 1* saves it in *Storage* and relays it to *Application 2*. *Analyser* uninterruptedly analyses data contained in *Storage* and sends its analysis also to *Application 2*. With all these inputs on hand, *Application 2* executes a certain algorithm and transmits the results to *GUI 1* and *GUI 2*, which, lastly, reveal them to their users. In this scenario, many actors play multiple roles. For instance, *User 1* is a data owner and a data viewer, and *Storage* is a data provider, a data consumer, and a big data framework.

We can see NIST's taxonomy as a technology-agnostic modelling language, and the figure above as a model designed with it. Technological choices are not indicated: *Storage* may be implemented by Apache Cassandra or HDFS, and *Analyser* may be a MapReduce or a Spark job. Some of the rules of this modelling language informally could be:

1. Every node has a name (e.g. "Analyser"), an icon, and labels—at least one.
2. Node names and node icons are freely chosen.
3. Allowed node labels are: data owner (DO), data viewer (DV), system orchestrator (SO), big data application provider (BDAP), big data framework provider (BDFP), data provider (DP), big data application (BDA), data consumer (DC), big data framework (BDF), infrastructure framework (IF), data platform framework (DPF), and processing framework (PF).
4. Since big data applications are data providers, a node labelled with BDA must also be labelled with DB. Similarly, a node labelled with IF, DPF, or PF must also be labelled with BDF.
5. An information flow is allowed only: (a) from a data owner to a data provider; and (b) from a data consumer to a data viewer.
6. A data flow is allowed only: (a) from a data provider to a big data application; (b) from a big data application to a data consumer or a big data framework; and (c) from a big data framework to a big data application or another big data framework.
7. A "provides" association is allowed only: (a) from a big data application provider to a big data application; and (b) from a big data framework provider to a big data framework.
8. A "uses" association is allowed only: (a) from a system orchestrator to a big data application; (b) from a data owner to a data provider; (c) from a big data application to a data provider or a big data framework; (d) from a big data framework to another big data framework; (e) etc.
9. Etc.

The Meta-Object Facility (MOF) is a standard of the Object Management Group (OMG) appropriate to define modelling languages rigorously. The famous Unified Modeling Language (UML) itself is specified with MOF. The case of UML is remarkable because UML has a profile mechanism that makes it possible for designers to specialise for a particular domain the meaning of all UML diagrams. In practice, language inventors have two options: either they begin from scratch and work directly with MOF, or they adapt UML to a subject of interest. Eclipse supports both methods. The Eclipse Modeling Framework (EMF) is a set of plugins for the Eclipse IDE that incorporates an implementation of MOF called Ecore. And the Eclipse project Papyrus supplies an implementation of UML based on Ecore.

In the previous section, we have given five advantages of model-driven DevOps. In the context of big data, there is more to say. Big data frameworks—infrastructure frameworks, data platform frameworks, and processing frameworks—are generally difficult to learn, configure, and manage, mainly because they involve scalable clusters of an unlimited number of computational resources. With model-driven DevOps, things become easier. Operators just declare in their models which technologies they want in the production environment, along with performance requirements. They let model-to-production tools install and configure the clusters automatically in a manner that satisfies the quality of service (QoS) requirements. The burden is partially borne by the tools.

Methodology Proposal

Now that we have clarified what is model-driven DevOps and the convenience thereof for big data, it is time to detail our software development methodology. The actors concerned by this methodology are big data application providers—developers and operators—and some of the system orchestrators—architects and project managers—since they are the only ones that actually construct, supervise, or monitor the construction of big data applications. In a nutshell, by following our approach, these actors easily experiment with big data systems' architectures thanks to a modelling language with which they give shape to their ideas. An architecture model, with all its big data technologies, is automatically and concretely reproducible onto a cloud infrastructure by model-to-production tools. So, they can compare what they envisioned and what they got. And every model change to improve performance or quality can be automatically propagated onto the cloud infrastructure too (continuous deployment). We divide the methodology into three scenarios that illustrate alternative ways to take advantage of model-driven DevOps for big data: architecture modelling, architecture analysis, and architecture experimentation.

Architecture Modelling

Nowadays, modelling has become a standard in software engineering. Whether it be for architectural decisions, documentation, code generation, or reverse engineering, today it is common for software specialists to draw models, and many industrial notations are at their disposal. Modelling is the cornerstone of our methodology. The modelling language of the DICE Consortium follows the philosophy of OMG's model-driven architecture (MDA): it includes three layers of abstraction: a platform-independent layer, a technology-specific layer, and a deployment-specific layer. With platform-independent concepts, architects describe a big data system in terms of computation node, source node, storage node, etc., without explicitly stating the underlying technologies. A DICE platform-independent model (DPIM) resembles a model done with NIST's taxonomy, except that the naming of concepts differ. Here is a partial conceptual correspondence:

DPIM concept	Corresponding concepts in NIST's taxonomy
Data-intensive application (DIA)	Big data system
Computation node	Processing framework and big data application
Source node	Data provider
Storage node	Data platform framework

Contrary to NIST's taxonomy, DPIM concepts are mutual exclusive. (DPIM source nodes do not provide persistence features; therefore, a storage node cannot be a source node.) Moreover, there is no DPIM concept for infrastructure frameworks because infrastructure is a low-level issue tackled at the deployment-specific layer. The full list of DPIM concepts will be explained in a subsequent chapter.

The DPIM model corresponds to the OMG MDA PIM layer and describes the behaviour of the application as a directed acyclic graph that expresses the dependencies between computations and data^[5].
—DICE Consortium

A DICE technology-specific model (DTSM) refines a DPIM by adopting a technology for each computation node, source node, storage node, etc. For example, an architect may choose Apache Cassandra or HDFS as one of his storage nodes. Again, DTSMs say nothing about the infrastructures in which these technologies will be deployed. It is the role of DICE deployment-specific models (DDSM) to specify them. DDSMs refine DTSMs with infrastructural choices. Here, the word *infrastructure* should be read as *infrastructure as a (programmable) service* (IaaS). It refers actually to the computing power of an infrastructure accessed over the Internet. Although the user does not see the underlying hardware, there is an application programming interface that enables him to programmatically create virtual machines or containers, select operating systems, and run software. A graphical user interface may allow him to carry out the same job interactively. Model-to-production tools rely on DDSMs and infrastructures' API to function.

Here are the steps of this scenario:

1. Draw an UML object diagram profiled with DPIM concepts to describe the components of a big data system.
2. Draw an UML activity diagram profiled with DPIM concepts to describe the actions of these components.
3. Refine the two previous diagram with DTSM concepts.
4. Draw an UML deployment diagram profiled with DDSM concepts to describe how the technologies will be deployed into infrastructures.
5. Generate scripts that use infrastructures' API to and run these scripts to obtain the production environment.

Architecture Analysis

Here are the steps of this scenario:

1. Draw an UML object diagram profiled with DPIM concepts to describe the components of a big data system.
2. Draw an UML activity diagram profiled with DPIM concepts to describe the actions of these components.
3. Draw a deployment diagram profiled with DPIM concepts to describe the deployment of the components into a simulated environment
4. Run a simulation.
5. Refine the three previous diagrams with DTSM concepts.
6. Run a simulation or a verification.
7. Run an optimisation to generate an optimised UML deployment diagram profiled with DDSM concepts.
8. Generate scripts that use infrastructures' API to and run these scripts to obtain the production environment.

Architecture Experimentation

Monitor and test the quality of the big data system in production and refactor of the models.

The DICE Methodology in the IDE

The DICE IDE is based on Eclipse, which is the de-facto standard for the creation of software engineering models based on the MDE approach. DICE customizes the Eclipse IDE with suitable plugins that integrate the execution of the different DICE tools in order to minimize learning curves and simplify adoption. Not all tools are integrated in the same way. Several integration patterns, focusing on the Eclipse plugin architecture, have been defined. They allow the implementation and incorporation of application features very quickly. DICE Tools are accessible through the DICE Tools menu.

The DICE IDE offers the ability to specify DIAs through UML models. From these models, the toolchain guides the Developer through the different phases of quality analysis, formal verification being one of them.

The IDE acts as the front-end of the methodology and plays a pivotal role in integrating the DICE tools of the framework. The DICE IDE can be used for any of the scenarios described in the methodology. The IDE is an integrated development environment tool for Model-driven engineering (MDE) where a Designer can create models at different levels (DPIM, DTSM and DDSM) to describe data-intensive applications and their underpinning technology stack.

The DICE IDE initially offers the ability to specify the data-intensive application through UML models stereotyped with DICE profiles. From these models, the tool-chain guides the developer through the different phases of quality analysis (e.g., simulation and/or formal verification), deployment, testing and acquisition of feedback data through monitoring data collection and successive data warehousing. Based on runtime data, an iterative quality enhancements tool-chain detects quality incidents and design anti-patterns. Feedbacks are then used to guide the Developer through cycles of iterative quality enhancements.

References

1. ISO/IEC (2015). "Big Data. Preliminary Report 2014" ISO. https://www.iso.org/files/live/sites/isoorg/files/developing_standards/docs/en/big_data_report-jtc1.pdf
2. NIST Big Data Public Working Group (2015-09). "NIST Big Data Interoperability Framework: Volume 1, Definitions. Final Version 1". NIST. doi:10.6028/NIST.SP.1500-1. https://bigdatawg.nist.gov/_uploadfiles/NISTSP.1500-1.pdf
3. NIST Big Data Public Working Group (2015-09). "NIST Big Data Interoperability Framework: Volume 2, Big Data Taxonomies. Final Version 1". NIST. doi:10.6028/NIST.SP.1500-2. https://bigdatawg.nist.gov/_uploadfiles/NISTSP.1500-2.pdf
4. NIST Big Data Public Working Group (2015-09). "NIST Big Data Interoperability Framework: Volume 6, Reference Architecture. Final Version 1". NIST. doi:10.6028/NIST.SP.1500-6. https://bigdatawg.nist.gov/_uploadfiles/NISTSP.1500-6.pdf
5. DICE Consortium (2015)."DICE: Quality-Driven Development of Data-Intensive Cloud Applications"<http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2014/11/mise15dice-position-paper.pdf>

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Methodology&oldid=3367866'

This page was last edited on 28 January 2018, at 09:52.

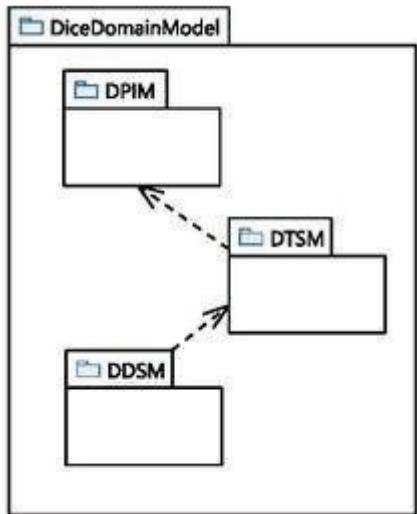
Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Introduction to Modelling

Contents

- [Introduction](#)
- [How the Profile Works](#)
- [References](#)

Introduction

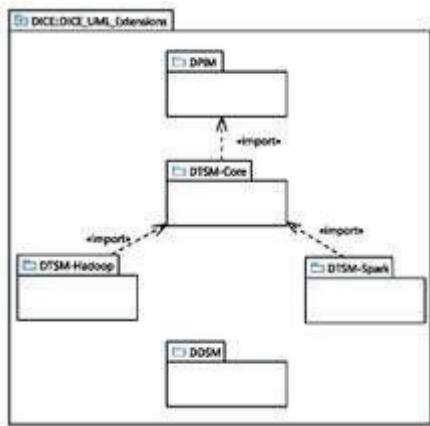


General View of DICE Profile

In the scope of this book, we introduce the DICE profile, its foundations and its architecture; further on, we outline how DICE-profiled models can be exploited in DevOps software development phases. The DICE profile has been structured to fit different abstraction levels (DPIM, DTSM, DDSM) similarly to the OMG Model-Driven Architecture^[1] (MDA) standard. For its construction, we have followed a guided process as recommended by state of the art works or building quality profiles. The DICE profile has been implemented and integrated within Papyrus UML^[2], a UML modelling tool based on the well-known Eclipse integrated development environment. The DICE domain models and the DICE profile are publicly available under an open source license in their corresponding repositories, namely the DICE-Models Repository and the DICE-Profiles Repository. In the future, we will focus on the continued validation of the DICE profile.

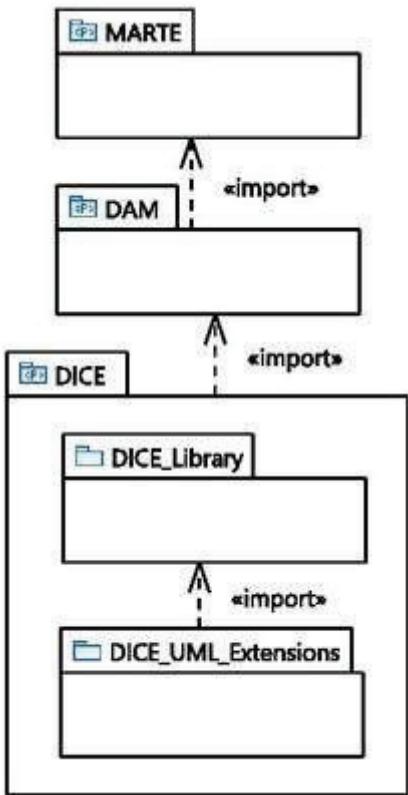
For constructing a technically correct high-quality UML profile that covers the necessary concepts according to data intensive applications and corresponding Big Data technologies, several steps need to be followed. First, conceptual models for each abstraction level, i.e. DPIM, DTSM and DDSM, are needed. We have carried out this step by carefully reviewing the abstract concepts for modeling data intensive applications. Hence, we have obtained the abstractions for the DPIM level, which then conform the DICE domain model at DPIM level (see [Platform-Independent Modelling](#)). Later, we have reviewed the different Big Data technologies addressed by DICE (e.g., Hadoop, Spark or Storm) and we have defined the abstractions of interest, consequently

obtaining the DICE domain model at DTSM level (see [Technology-Specific Modelling](#)). Finally, a Big Data application is deployed over a specific cloud architecture, whose deployment is captured by a DICE deployment model at DDSM level (see [Deployment-Specific Modelling](#)).



DICE UML Extensions

As a second step, we realized the need of introducing fresh concepts for quality assessment since the DICE Profile, at DPIM and DTSM domain model, initially just offers concepts for describing an architectural view. Therefore, we searched in the literature for existing UML profiles that leverage quality concerns, and decided to incorporate MARTE^[3] and DAM^[4]. Hence, the DICE Profile has deep roots on these two profiles. Our task was to select from the domain models of MARTE and DAM those metaclasses of interest for supporting our specific needs on assessment. We studied how to integrate such metaclasses and the already developed DPIM domain model. Consequently we gained a final domain model which integrates all needed features: applications abstractions at DPIM level and behavioral abstractions for quality assessment.



Structure of the DICE Profile

As a third step, we faced the technical details of a profile construction. We needed to map the concepts of the DICE profile, at DPIM, DTMS and DDSM domain models, into proper UML profile constructors, i.e., stereotypes and tags. In particular, we have designed: (i) the DICE Library, containing data intensive applications specific types; and (ii) the DICE UML Extensions (stereotypes and tags). The objective was to introduce a small yet comprehensive set of stereotypes for the software designer. Finally, as a fourth and last

step, we conducted a DICE UML profile assessment by identifying a set of requirements based on three case studies from different application domains: fraud detection (see [Fraud Detection](#)), vessel traffic management (see [Maritime Operations](#)), and acquisition of news from social sensors (see [News and Media](#)). We checked if the requirements were met by the profile and, if a requirement was not met, we went back to the previous step in order to refine it. Therefore, we followed an iterative process for the profile definition.

How the Profile Works

While the annotated UML model is useful for the engineer to specify both the workflow of the DIA and its data characteristics, it is not suitable for an assessment of its performance requirements. Following the model-driven engineering paradigm, our goal is to define a quality-driven framework for developing DIA applications leveraging Big Data technologies. A key asset of DICE is the so called DICE profile, which offers the ability to design DIA using UML and a set of additional stereotypes to characterize specific DIA features. DICE-profiled models are the cornerstone of the DICE framework, since they are exploited by the DICE tool-chain to guide developers through the whole DIA lifecycle (e.g., development, quality analysis, deployment, testing, monitoring, etc.).

In this chapter, we have presented the DICE profile, its foundations and its architecture; and we have outlined how DICE-profiled models can be exploited in further software development phases. The DICE profile has deep roots on other two profiles, namely MARTE and DAM, and has been structured to fit different abstraction levels (DPIM, DTSM, DDSM) similarly to the MDA standard. For its construction, we have followed a guided process as recommended by state of the art works for building quality profiles. In the following sections, we give the details of the DICE Profile at each abstraction level.

References

1. Object Management Group (OMG). "Model-Driven Architecture Specification and Standardisation" <http://www.omg.org/mda/>
2. The Eclipse Foundation (2010). *A slide-ware tutorial on Papyrus usage for starters* https://eclipse.org/papyrus/users/Tutorials/resources/TutorialOnPapyrusUSE_d20101001.pdf
3. Object Management Group (OMG) (2011). *UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems*. 1.1. <http://www.omg.org/spec/MARTE/1.1/>.
4. Bernardi, Simona; J. Merseguer D.C. Petriu (2011). "A dependability profile within MARTE.". *Software and Systems Modeling* **10** (3): 313-336. doi:<https://doi.org/10.1007/s10270-009-0128-1>?LI=true <https://link.springer.com/article/10.1007%2Fs10270-009-0128-1?LI=true>

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Introduction_to_Modelling&oldid=3367979'

This page was last edited on 28 January 2018, at 15:52.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Platform-Independent Modelling

Contents

Introduction

DPIM Profile

DPIM Example: The Maritime Operations Case Study

Conclusion

References

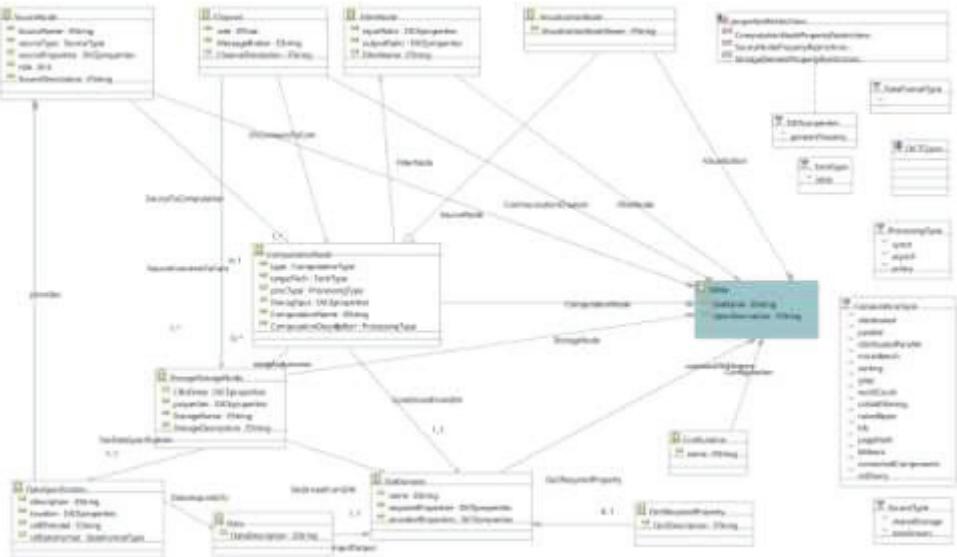
Introduction

DICE provides Software Architects with a set of core concepts, at the DPIM layer, to specify the fundamental architecture elements that constitute a Data-Intensive Application (DIA), i.e., during the DIA Design phase. Designers may use the identified core architecture elements to quickly put together the structural view of their Big-Data application, highlighting and tackling concerns such as data flow and essential high-level processing properties (e.g., rate, properties provided and required by every component, etc.) as well as key data processing needs (e.g., batch, streaming, etc.).

DPIM Profile

DPIM includes all concepts that are relevant to structure a DIA. At the DPIM level we define the high level topology of the application and its QoS requirements. Elements of the DPIM meta-model fall into two categories:

1. Active DIA elements, which process the data, such as computational nodes ;
2. Passive DIA elements, which stores and visualize the data, such as the storage nodes ;



Picture of the DICE DPIM Profile (Meta-model)

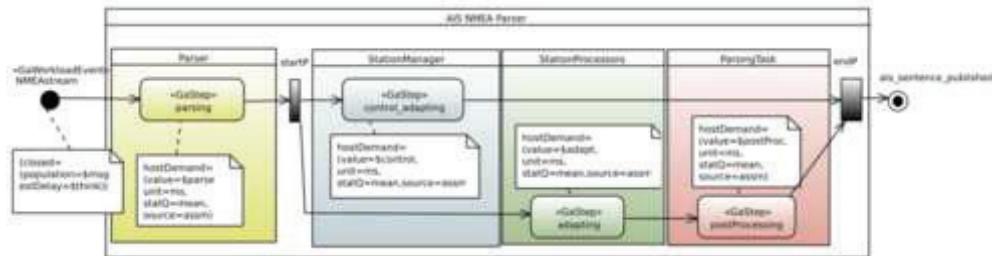
More in particular, the DICE DPIM Profile meta-model shows that DIA elements are essentially aggregates of two sets of components. Firstly, the "ComputationNode", which is basically responsible for carrying out computational task like map, or reduce in MapReduce. One of important attributes of ComputationNode is "computationType" that shows the processing type of big-data i.e, batch processing or stream processing. The ComputationNode itself, further specializes into "SourceNode" and "Visualization" nodes. The role of the SourceNode is to provide data for processing. In other words, the SourceNode represents the source of data which are coming into application in order to being processed. The attribute "sourceType" further specifies the characteristics of source. The ultimate goal of a big-data application is to process the data that have high volume and velocity. So the SourceNode, and ComputationNode are in DPIM since there are the essential part of each and every DIA. The sourceNode is the entry point of data into the application and the Computation is where data would be processed. Visualization here means to visualize the data to represent the knowledge more intuitively and effectively by using different graphs which are computed through Data-Intensive means. Even though, the visualization of big-data itself could be done by a separate application, but here we considered visualization as specification of ComputationNode since ultimately the visualization is a data-intensive computation task. Another element which is also specification of ComputationNode is the FilterNode. Its role is to do any type of pre-processing and post-processing of data if needed.

The second key element in the DICE profile is the "StorageNode". As its name may suggest, the StorageNode represents the element which is responsible to store the data, either for long or short term. Moreover, it is associated with "Channel" that represents the communication channel in the application. The specification of Channel also shows the restrictions and constraints of a channel. It also specifies the characteristics related to transformation of data like information rate and taps. The concept of StorageNode in DPIM mainly corresponds with the "database" in the model. In some cases, it could also be a "filesystem". The channel in DPIM is a representation of "Governance and data Integration" in which mainly includes the technologies responsible for transferring the data, like message broker systems. The other elements in the model are "DataSpecification" and "QoSRequiredProperty", which are annotation stubs for specification the type and format of data and the QoS for system and its evaluation respectively. These annotations are inherited from MODACloudML^[1]. Appendix A^[2] specifies the DICE Profile with greater detail. Table 1 summarizes the current list of stereotypes of the DICE Profile for the DPIM level.

Stereotype	Description (This stereotype is for model elements representing...)
DpimComputationNode	DIA components with computation throughput, type of data processing, and maybe expected target technology
DpimFilterNode	Filter nodes that extend general DpimComputationNode with input and output ratios.
DpimSourceNode	DIA components with a given storage volume, type of generated data, and data generation rate.
DpimStorageNode	DIA component with resource multiplicitytype of stored data, and speed in terms of maximum operations rate.
DpimChannel	Connectors that have a maximum speed and that are subject to failures and propagation of errors.
DpimScenario	An execution scenario of the DIA, which defines the quality properties of interest and the scenario quality requirements.

DPIM Example: The Maritime Operations Case Study

In this section, we describe a UML-based design (i.e., Activity Diagram) that is annotated using the DPIM profile. In particular, input parameters are assigned to the mean durations of the action steps (i.e., hostDemand tagged-values) and to the data stream arrival rate (i.e., arrivalRate tagged-value). We show the modelling of a portion of the [Maritime Operations case study](#).



Profiled Activity Diagram

As previously explained in [Introduction to Modelling](#), the DPIM profile rely on the standard MARTE and DAM profiles. This is because DAM is a profile specialized in dependability and reliability analysis, and MARTE offers the GQAM sub-profile, a complete framework for quantitative analysis. Therefore, they matches perfectly to our purposes: the quality assessment of data intensive applications. Moreover, MARTE offers the NFPs and VSL sub-profiles. The NFP sub-profile aims to describe the non-functional properties of a system, performance in our case. The latter, the VSL sub-profile, provides a concrete textual language for specifying the values of metrics, constraints, properties, and parameters related to performance. VSL expressions are used in DPIM-profiled models with two main goals: (i) to specify the input parameters of the model and (ii) to specify the performance metric(s) that will be computed for the model (i.e., the output results). An example of VSL expression for a host demand tagged value of type NFP_Duration is:

expr=\$parse (1), unit=ms (2), statQ=mean (3), source=est (4)

This expression specifies that the parsing step in (yellow box in the image) demands \$parse (1) milliseconds (2) of processing time, whose mean value (3) will be obtained from an estimation in the real system (4). \$parse is a variable that can be set with concrete values during the analysis of the model.

Conclusion

DICE UML-based application modelling heavily rotates around DPIM, a new refined UML profile to specify the fundamental architecture elements that constitute a Data-Intensive Application (DIA).

References

1. MODAClouds (2016).MODACloudML Development - Initial Version (Report). http://www.modaclouds.eu/wp-content/uploads/2012/09/MODAClouds_D4.2.1_MODACloudMLDevelopmentInitialVersion.pdf.
 2. The DICE Consortium (2016).Design and Quality Abstractions - Initial Version (Report). http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2016/02/D2.1_Design-and-quality-abstractions-Initial-version.pdf
-

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Platform-Independent_Modelling&oldid=3367972'

This page was last edited on 28 January 2018, at 15:41.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Technology-Specific Modelling

Contents

Introduction

DTSM Modelling Explained: The Apache Storm Example

Storm Concepts

Storm Profile

Introduction

When all essential architecture elements are in place, by means of architectural reasoning in the DPIM layer it could be made available ad-hoc model transformations that parse DPIM models and produce equipotent DTSM models where the specified data processing needs are exploded (if possible) into a possible configuration using appropriate technologies (e.g., Spark for streaming or Hadoop for batch). At this layer it should be provided for architects and developers several key technological framework packages that can evaluate possible alternatives for Technological Mapping and Logical Implementation, that is, selecting the technological frameworks that map well with the problem at hand and implementing the needed processing logic for that framework. Once designers choose the appropriate technological alternative, DICE will provide model transformations that instantiate the alternative (if available) e.g., by instantiating pre-made, ad-hoc packages that contain: (a) framework elements needed to "link" the Data-Intensive application logic (e.g., through inheritance); (b) framework elements that contain (optional) configuration details (c) framework elements that represent deployable entities and nodes (e.g., Master Nodes and Resource Managers for Hadoop Map-Reduce). Software Architects proceed by filling out any wanted configuration details to run the chosen frameworks, probably in collaboration with Infrastructure Engineers.

DTSM Modelling Explained: The Apache Storm Example

The Data-Intensive Technology Specific profile (DTSM) includes several technology-specific sub-profiles, including the DTSM::Storm Profile, the DTSM::Hadoop Profile, and the refinement of the DICE::DICE_Library. The Hadoop and Storm Profiles fully incorporate quality and reliability aspects. Other DTSM profiles, such as the Spark profile, are currently under further experimentation and will be finalized in the near future. In summary the Apache Storm profile core elements, stereotypes, and tagged values are defined in the following file.

Storm Concepts

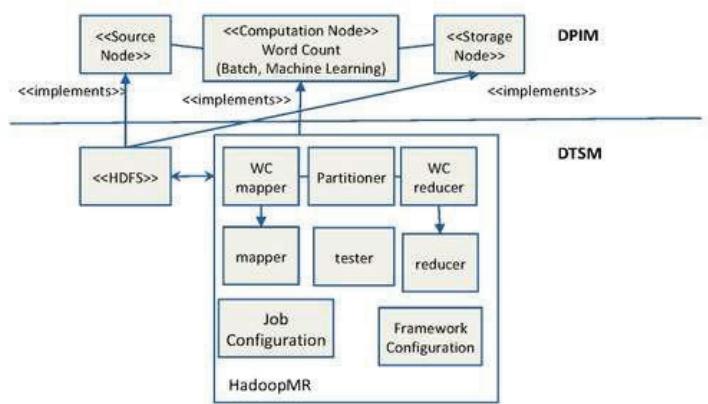
Storm concepts that impact in performance

#	Concept	Meaning
1.	<i>Spout</i> (task)	Source of information
2.	<i>Emission rate</i>	Number of tuples per unit of time produced by a spout
3.	<i>Bolt</i> (task)	Data elaboration and production of results
4.	<i>Execution time</i>	Time required for producing an output by a bolt
5.	<i>Ratio</i>	Number of tuples required or produced
6.	<i>Asynchronous policy</i>	The bolt waits until it receives tuples from any of the incoming streams
7.	<i>Synchronous policy</i>	The bolt waits until it receives tuples from all the incoming streams
8.	<i>Parallelism</i>	Number of concurrent threads per task
9.	<i>Grouping</i>	Tuple propagation policy (shuffle/all/field/global)

Storm is a distributed real-time computation system for processing large volumes of high-velocity data. A Storm application is usually designed as a directed acyclic graph (DAG) whose nodes are the points where the information is generated or processed, and the edges define the connections for the transmission of data from one node to another. Two classes of nodes are considered in the topology. On the one hand, spouts are sources of information that inject streams of data into the topology at a certain emission rate. On the other hand, bolts consume input data and produce results which, in turn, are emitted towards other bolts of the topology.

A bolt represents a generic processing component that requires n tuples for producing m results. This asymmetry is captured by the ratio m/n. Spouts and bolts take a certain amount of time for processing a single tuple.

Besides, different synchronization policies shall be considered. A bolt receiving messages from two or more sources can select to either 1) progress if at least a tuple from any of the sources is available (asynchronously), or 2) wait for a message from all the sources (synchronously).



DPIM-to-DTSM Transformation Example: Apache Hadoop MapReduce

A Storm application is also configurable by the parallelism of the nodes and the stream grouping. The parallelism specifies the number of concurrent tasks executing the same type of node (spout or bolt). Usually, each task corresponds to one thread of execution. The stream grouping determines the way a message is propagated to and handled by the receiving nodes. By default, a message is broadcast to every successor of the current node. Once the message arrives to a bolt, it is redirected randomly to any of the multiple internal threads (shuffle), copied to all of them (all) or to a specific subset of threads according to some criteria (i.e., field, global, etc.).

As per its own definition, the DTSM Profile includes a list of stereotypes that addresses the main concepts of the Apache Storm technology. In particular, we stress those concepts that directly impact on the performance of the system. Consequently, these parameters are essential for the performance analysis of the Storm applications and are useful for the DICE Simulation, Verification and Optimization tools.

In addition, a Storm framework is highly configurable by various parameters that will influence the final performance of the application. These concepts are converted into stereotypes and tags, which are the extension mechanisms offered by UML. Therefore, we devised a new UML profile for Storm. In our case, we are extending UML with the Storm concepts. The stereotypes and annotations for Storm are based on MARTE, DAM, the DICE::DPIM and Core profiles. The DICE::DTSM::Storm profile provides genuine stereotypes.

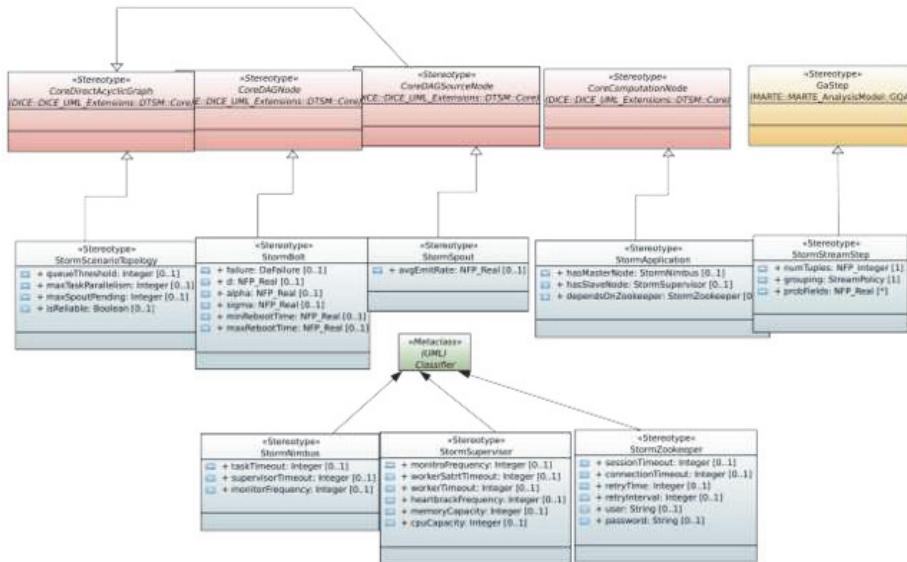
Storm Profile

Spouts and bolts have independent stereotypes because they are conceptually different, but <<StormSpout>> and <<StormBolt>> inherit from MARTE::GQAM::GaStep stereotype via the DTSM::Core::Core-DAG-Node and CoreDAGSourceNodes since they are computational steps. Moreover, they share the parallelism, or number of concurrent threads executing the task, which is specified by the tag parallelism.

On the other hand, the spouts add the tag avgEmitRate, which represents the emission rate at which the spout produces tuples. Finally, the bolts use the hostDemand tag from GaStep for defining the task execution time. The time needed to process a single tuple can also be expressed through the alpha tag in case that the temporal units are not specified. The minimum and the maximum time to recover from a failure is denoted by the minRebootTime and maxRebootTime tags.

The Storm concept of stream is captured by the stereotype <<StormStreamStep>>. This stereotype also inherits from MARTE::GQAM::GaStep stereotype, which enables to apply it to the control flow arcs of the UML activity diagram. This stereotype has two tags, grouping and numTuples that match the grouping and ratio concepts in Storm, respectively. The type of grouping is StreamPolicy, an enumeration type of the package Basic_DICE_Types that has the values all, shuffle, field, global for indicating the message passing policy. The ratio of m/n of a bolt can be expressed either through the attribute sigma in the bolt stereotype, or by specifying the incoming and outgoing numples of a bolt via the stream stereotype.

Finally, the <<StormScenarioTopology>> stereotype is introduced for defining contextual execution information of a Storm application. That is, the reliability of the application or the buffer size of each task for exchanging messages. All these stereotypes are summarized in the following picture, which describes the DTSM::Storm profile for UML-based environments.



Description of DTSM::Storm profile

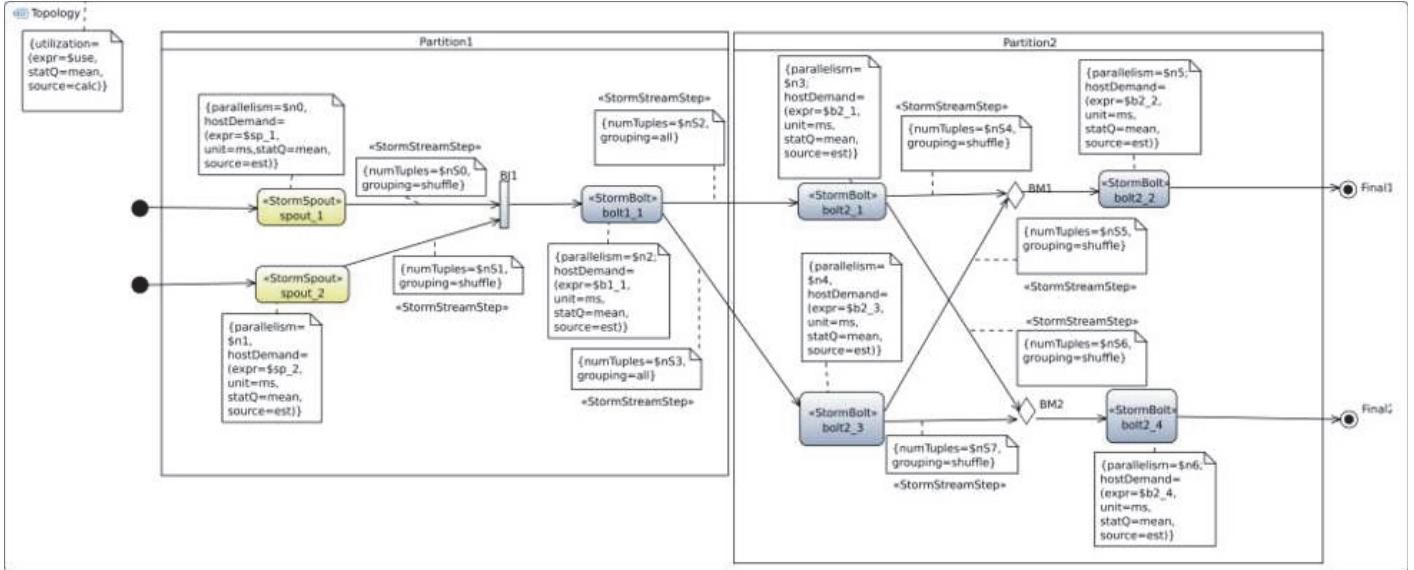
As previously explained in [Introduction to Modelling the DTSM profiles](#), and particularly the DTSM::Storm profile, rely on the standard MARTE and DAM profiles. This is because DAM is a profile specialized in dependability and reliability analysis, and MARTE offers the GQAM sub-profile, a complete framework for quantitative analysis. Therefore, they matches perfectly to our purposes: the quality assessment of data intensive applications. Moreover, MARTE offers the NFPs and VSL sub-profiles. The NFP sub-profile aims to describe the non-functional properties of a system, performance in our case. The latter, the VSL sub-profile, provides a concrete textual language for specifying the values of metrics, constraints, properties, and parameters related to performance. VSL expressions are used in Storm-profiled models with two main goals: (i) to specify the input parameters of the model and (ii) to specify the performance metric(s) that will be computed for the model (i.e., the output results). An example of VSL expression for a host demand tagged value of type NFP_Duration is:

expr=\$b_1 (1), unit=ms (2), statQ=mean (3), source=est (4)

This expression specifies that bolt_1 demands \$b_1 (1) milliseconds (2) of processing time, whose mean value (3) will be obtained from an estimation in the real system (4). \$b_1 is a variable that can be set with concrete values during the analysis of the model. Another VSL interesting expression is the definition of the performance metric to be calculated, the utilization in the image of next example:

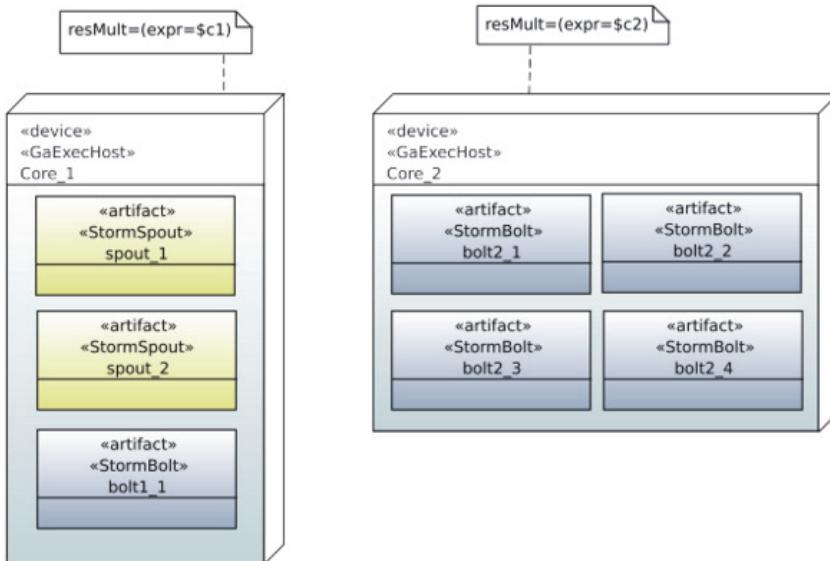
expr=\$use (1), statQ=mean (2), source=calc (3)

This expression specifies that we want to calculate (3) the utilization, as a percentage of time, of the whole system or a specific resource, whose mean value (2) will be assigned to variable \$use (1). Such value is obtained by the evaluation of a performance model associated to this Storm-profiled UML diagram. The rest of picture corresponds to a UML activity diagram that represents the Storm DAG. Every UML element is stereotyped with a DTSM::Storm stereotype matching one of the concepts presented in the previous table. All the configuration parameters of the profiled elements (\$_) are variables.



Activity Diagram of Storm example

Nodes in the UML activity diagram are grouped by partitions (e.g., Partition1 and Partition2). Each partition is mapped to a computational resource in the UML deployment diagram following the scheduling policy defined for the topology. Next picture shows the deployment diagram, which complements the previous activity diagram. Each computational resource is stereotyped as GaExecHost and defines its resource multiplicity, i.e., number of cores. The deployment also allows one to know which messages exchanged by tasks can introduce network delays, i.e., tuples exchanged between cores in different physical machines, which is of importance for the eventual performance model. Therefore, we use the GaCommHost stereotype. Both stereotypes are inherited from MARTE GQAM.



Deployment Diagram of Storm example

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data&oldid=3367980

This page was last edited on 28 January 2018, at 15:52.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Deployment-Specific Modelling

Contents

Introduction

Technical Overview

DDSM UML Deployment Profile

UML Deployment Modelling: The ~~Wi~~kiStats Example

Conclusion

Introduction

DIAs and the Big Data assets these manipulate are key to industrial innovation. However going data-intensive requires much more than not only in design, but also in system/infrastructure configuration and deployment - these still happen via heavy manual fine-tuning and trial-and-error. We outline abstractions and automations that support data-intensive deployment and operation in an automated DevOps fashion, featuring Infrastructure-as-Code and TOSCA.

Concerning Infrastructure-as-Code and TOSCA in the specific, they reflect the DevOps tactic to adopt source-code practicals in infrastructure design as well. More in particular, infrastructure-as-code envisions the definition, versioning, evaluation, testing, etc. of source-code for infrastructural designs just as application code is defined, versioned, evaluated, and tested. TOSCA, is the OASIS standard definition language for infrastructure-as-code and stands for "Topology and Orchestration Specification for Cloud Applications".

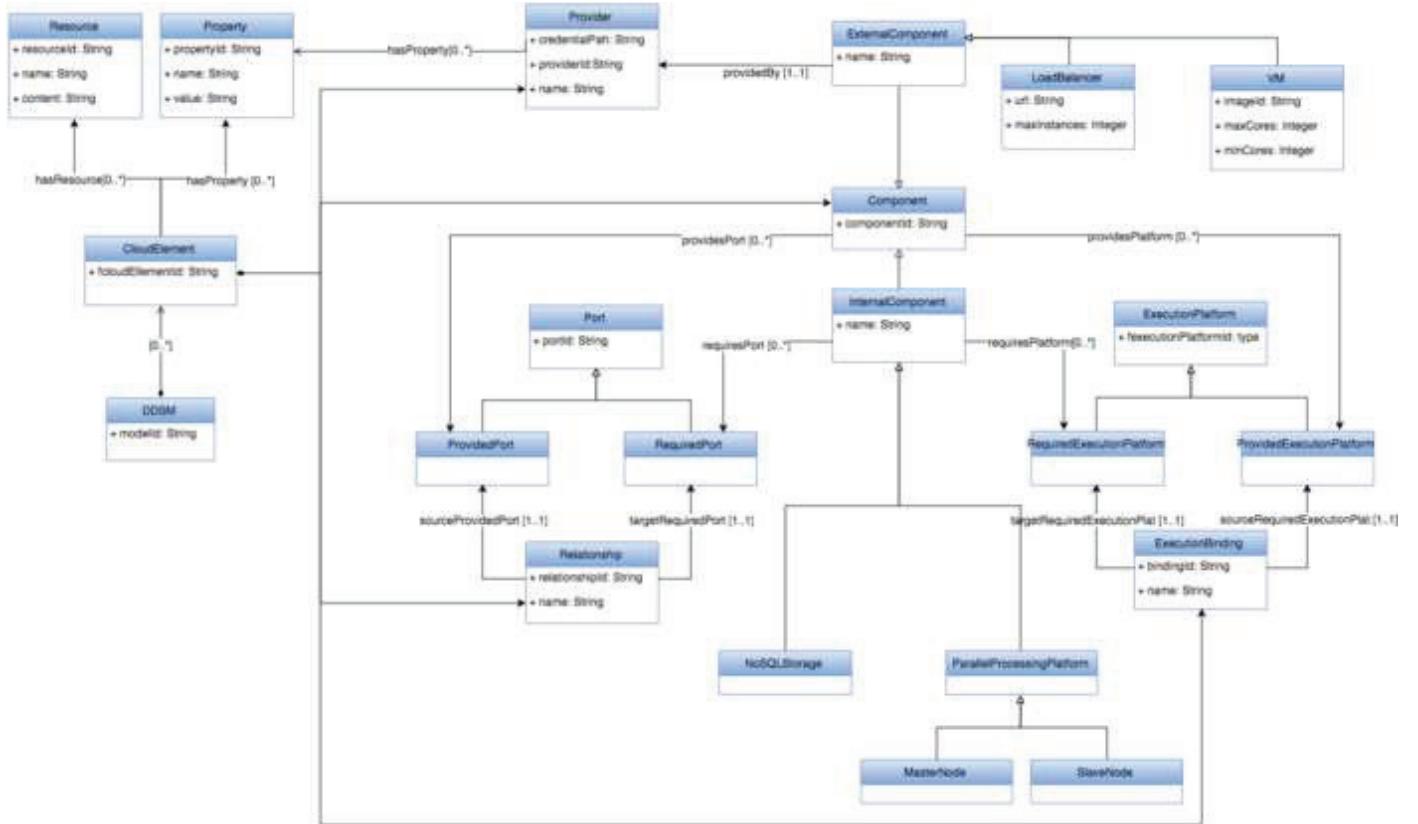
The DDSM allows to express the deployment of DIAs on the Cloud, using UML.

Technical Overview

On one hand, IaC is a typical DevOps tactic that offers standard ways to specify the deployment infrastructure and support operations concerns using human-readable notations. The IaC paradigm features: (a) domain-specific languages (DSLs) for Cloud application specification such as TOSCA, i.e., the "Topology and Orchestration Specification for Cloud Applications" standard, to program the way a Cloud application should be deployed; (b) specific executors, called orchestrators, that consume IaC blueprints and automate the deployment based on those IaC blueprints.

On the other hand, DDSM framework is a UML-based modeling framework based on the MODAClouds4DICER meta-model, which is a transposition and an extension of the MODACloudsML meta-model adapted for the intents and purposes of data intensive deployment. MODACloudsML is a language that allows to model the provisioning and deployment of multi-cloud applications exploiting a component-based approach. The main motivation behind the adoption of such a language on top of TOSCA is that we want to make the design methodology TOSCA-independent, in such a way that the designer have not to be a TOSCA-expert, nor even to be aware about TOSCA, but he should just follow the proposed methodology. Moreover the MODACloudsML language has basically the same purpose of the TOSCA standard, but it exhibits a higher level of abstraction and so results in being more user friendly. The below Figure shows an extract of the MODAClouds4DICER meta-model. The main concepts are inherited directly from MODACloudsML. A MODACloudsML model is a set of Components which can be owned by a Cloud provider ExternalComponents or by the application provider InternalComponents. A Component can be either an application, a platform or a physical host. While an ExternalComponent can just provide Ports and ExecutionPlatforms, an InternalComponent can also require them, since it is controlled

by the application provider. Ports and ExecutionPlatforms serve as a way to connect Components to each other. ProvidedPorts and RequiredPorts can be linked by mean of the concept of Relationship, while ProvidedExecutionPlatforms and RequiredExecutionPlatforms can be linked by mean of the concept of ExecutionBinding. The latter could be seen as a particular type of relationship between two Components which tells that one of them is executing the other



ModaClouds4TOSCA or DDSM Notation.

MODACloudsML has been adapted extending elements in order to capture data intensive specific concepts, e.g. systems that are usually exploited by data intensive applications such as NoSQLStorage solutions and ParallelProcessingPlatforms, which are typically composed of a MasterNode and one or many SlaveNodes.

DDSM UML Deployment Profile

Stemming from the previous technical overview, in the following we elaborate on essential DDSM stereotypes, which are reported in the below Table.

DDSM main stereotypes

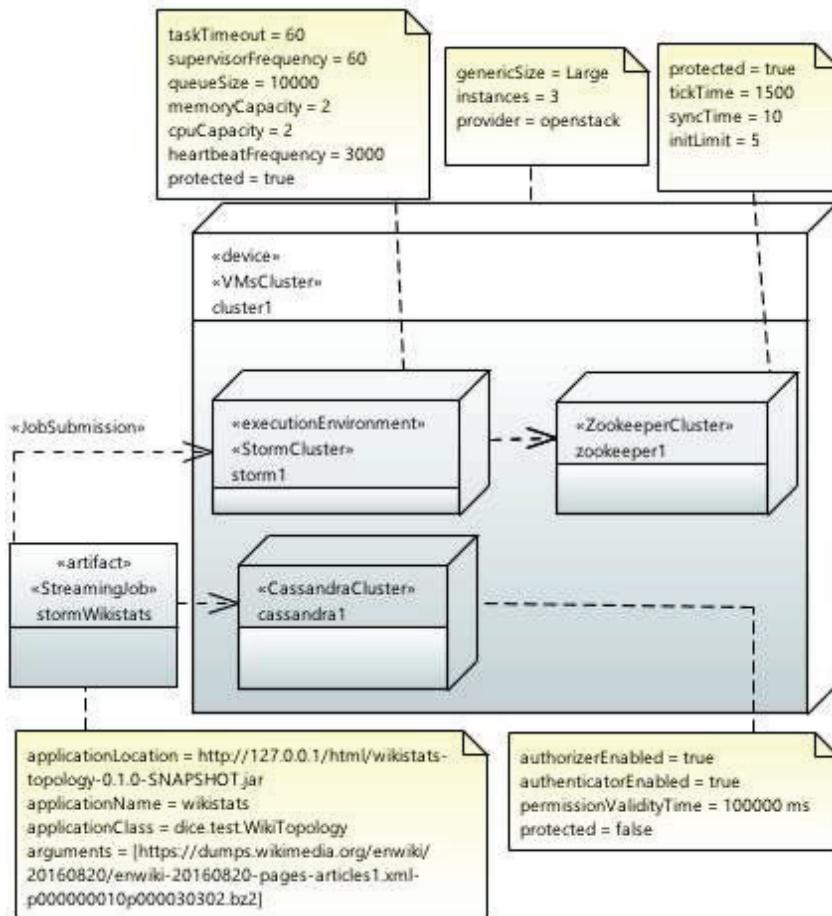
#	Stereotype	Meaning
1.	<i>InternalNode</i>	Service that are managed and deployed by the application owner
2.	<i>ExternalNode</i>	Service that are managed and deployed by the third-party provider
3.	<i>VMsCluster</i>	A cluster of virtual machines
4.	<i>PeerToPeerPlatform</i>	A data-intensive platform operating according to the peer-to-peer style
5.	<i>MasterSlavePlatform</i>	A data-intensive platform operating according to the master-slave style
6.	<i>StormCluster</i>	An instance of a Storm cluster
7.	<i>CassandraCluster</i>	An instance of a Cassandra cluster
8.	<i>BigDataJob</i>	The actual DIA to be executed
9.	<i>JobSubmission</i>	Deployment association between a BigDataJob and its corresponding execution environment

- DDSM distinguishes between *InternalNode*, or services that are managed and deployed by the application owner and *ExternalNode* that are owned and managed by a third-party provider (see the *providerType* property of the *ExternalNode* stereotype). Both the *InternalNode* and *ExternalNode* stereotypes extend the the UML meta-class *Node*.
- *VMsCluster* stereotype is defined as a specialisation of *ExternalNode*, as renting computational resources such as virtual machines is one of the main services (so called Infrastructure-as-a-Service) offered by Cloud providers. *VMsCluster* also extends the *Device* UML meta-class, since a cluster of VMs logically represents a single computational resource with processing capabilities, upon which applications and services may be deployed for execution. A *VMsCluster* has an *instances* property representing its replication factor i.e., the number of VMs composing the cluster. VMs in a cluster are all of the same size (in terms of amount of memory/number of cores, clock frequency), which can be defined by means of the *MSize* enumeration.
- Alternatively the user can specify lower and upper bounds for the VMs' characteristics (e.g. *minCore/maxCore*, *minRam/maxRam*), assuming the employed Cloud orchestrator is then able to decide the optimal Cloud offer, according to some criteria, that matches the specified bounds. The *VMsCluster* stereotype is fundamental towards providing DDSM users with the right level of abstraction, so that they can model the deployment of DIAs, without having to deal with the complexity exposed by the underlying distributed computing infrastructure. In fact, an user just has to model her clusters of VMs as stereotyped Devices that can have nested *InternalNodes* representing the hosted distributed platforms. Furthermore, a specific OCL constraint imposes that each *InternalNode* must be contained into a Device holding the *VMsCluster* stereotype, since by definition an *InternalNode* have to be deployed and managed by the application provider which thus has to dispose the necessary hosting resources.
- We then define DIA-specific deployment abstractions, i.e. the *PeerToPeerPlatform* *MasterSlavePlatform* stereotypes, as further specialisations of *InternalNode*. These two stereotypes basically allow the modelling language to capture the key differences between the two general type of distributed architectures. For instance the *MasterSlavePlatform* stereotype allows to indicate a dedicated host for the master node, since it might require more computational resources. By extending our deployment abstractions, we implemented a set of technology modelling elements (*StormCluster*, *CassandraCluster*, etc.), one for each technology we support. DIA execution engines (e.g. Spark or Storm) also extend UML *ExecutionEnvironment*, so to distinguish those platforms DIA jobs can be submitted to. Each technology element allows to model deployment aspects that are specific to a given technology such as platform specific configuration parameters or dependencies on other technologies, that are enforced by means of OCL constraints in the case they are mandatory.
- The *BigDataJob* stereotype represents the actual application that can be submitted for execution to any of the available execution engine. It is defined as a specialisation of UML *Artefact*, since it actually corresponds to the DIA executable artefact. It allows to specify job-specific information, for instance the *artifactUrl* from which the application executable can be retrieved.

- The JobSubmission stereotype, which extends UML Deployment, is used to specify additional deployment options of a DIA. For instance, it allows to specify job scheduling options, such as how many times it has to be submitted and the time interval between two subsequent submissions. In this way the same DIA job can be deployed in multiple instances using different deployment options. An additional OCL constraint requires each BigDataJob to be connected by mean of JobSubmission to a UML ExecutionEnvironment which holds a stereotype extending one between the MasterSlavePlatform or the PeerToPeerPlatform stereotypes.

UML Deployment Modelling: The Wikistats Example

We showcase the defined profile by applying it to model the deployment of a simple DIA that we called Wikistats, a streaming application which processes Wikimedia articles to elicit statistics on their contents and structure. The application features Apache Storm as a stream processing engine and uses Apache Cassandra as storage technology. Wikistats is a simple example of a DIA needing multiple, heterogeneous, distributed platforms such as Storm and Cassandra. Moreover Storm depends on Apache Zookeeper. The Wikistats application itself is a Storm application (a streaming job) packaged in a deployable artefact. The below Figure shows the DDSM for the Wikistats example.



DDSM of the Wikistats example.

In this specific example scenario, all the necessary platforms are deployed within the same cluster of 2 large-sized VMs from an OpenStack installation. Each of the required platform elements is modelled as a Node annotated with a corresponding technology specific stereotype. In particular Storm is modelled as an ExecutionEnvironment, as it is the application engine that executes the actual Wikistats application code. At this point, fine tuning of the Cloud infrastructure and of the various platforms is the key aspect supported by DDSM. The technology stereotypes allow to configure each platform in such a way to easily and quickly test different configurations over multiple deployments and enabling the continuous architecting of DIAs. The dependency of Storm on Zookeeper is enforced via the previously discussed OCL constraints library which comes automatically installed within the DDSM profile. The deployment of the Wikistats application is modelled as an Artefact annotated with the BigDataJob stereotype and linked with the StormCluster element using a Deployment dependency stereotyped as a JobSubmission. Finally BigDataJob and JobSubmission can be used to elaborate details about the Wikistats job and how it is scheduled.

Conclusion

DICE UML-based deployment modelling heavily rotates around DDSM, a refined UML profile to specify Infrastructure-as-Code using simple UML Deployment Diagrams stereotyped with appropriate data-intensive augmentations.

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Deployment-Specific_Modelling&oldid=3369233'

This page was last edited on 29 January 2018, at 12:58.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Quality Verification

Contents

Introduction

Motivations

Existing solutions

How the tool works

The verification process

The architecture

Open challenges

Application domain

Storm

Spark

A test case - DigitalPebble web crawler

Conclusion

Introduction

The analysis of system correctness is fundamental to produce systems whose runtime behavior is guaranteed to be correct. However, the notion of correctness is general and needs to be refined to suit the specific scenario that embeds such a system. To this end, appropriate criteria has to be considered to define what correctness is, according to the kind of system that is taken into account and to the user requirements that the implemented system should exhibit at runtime. Verification in DICE aims to define a possible meaning of correctness for DIAs and to provide tools supporting the formal analysis.

The tool that is available in DICE for the safety analysis of DIAs is **D-VerT**. Safety verification in DICE is performed to check, in the DIA model, the reachability of unwanted configurations, i.e., a malfunction which consists of behaviors that do not conform to some non-functional requirements specified by the Quality Engineer

Motivations

Verification is intended to focus on the analysis of the effect of an incorrect design of timing constraints which might cause a latency in the processing of data. Therefore, the causes of the undesired behaviors that reflect in practice on the outcome of the verification can be inferred by the designers using the analysis tool.

The design of timing constraints is an aspect of the software design which is quite relevant in applications such as those considered in DICE. The unforeseen delay can actually lead the system to incorrect behaviors that might appear at runtime in various form depending on the kind of application under development. Underestimating the computational power of nodes affects the time that is needed by the node to process the input data, which, in turn, might have an effect on the whole time span of the application, as the processing delay might trigger a chain reaction. For instance, in a streaming application, the slowness of a node might cause accumulation of messages in the queues and lead possibly to memory saturation, if no procedures take action to deal with the anomaly. In a batch application, an unpredicted delay might affect the total processing time and alter the application behavior which, in turn, violates the Service Level Agreement with the customers.

Existing solutions

To the best of authors' knowledge, there are no available tools offering such peculiarities that are specifically developed for the DICE reference technologies

How the tool works

DICE adopts model checking techniques to verify DIA specified in DTSM diagrams. A verification problem is specified by a formal description of the DIA (an annotated DTSM models that contain suitable information about the timing features of the application undergoing analysis) and a logical formula representing the property that its executions must satisfy

The verification process in DICE relies on a fully automatic procedure that is based on dense-time temporal logic and it is realized in accordance with the bounded model-checking approach. It is designed to be carried out in an agile way: the DIA designer performs verification by using a lightweight approach. More precisely, **D-VerT** fosters an approach whereby formal verification is launched through interfaces that hide the complexity of the underlying models and engines. These interfaces allow the user to easily produce the formal model to be verified along with the properties to be checked and eliminates the need for experts of the formal verification techniques.

The outcome of the verification is used for refining the model of the application at design time, in case anomalies are detected by **D-VerT** (see use case).

The verification process

The verification process consists of the following steps. The designer draws a class diagram in the DICE IDE representing the DTSM model of the DIA and, afterward, provides all the annotations required for the analysis, based on the selected technology that she employs to implement the application. When the user starts the analysis, the annotated DTSM model is converted into a formal model that represents the abstracted behavior of the application at runtime. Based on the kind of system to implement - either a Storm application or a Spark application - the tool selects the class of properties to verify and performs the analysis. Finally, when the outcome is available, the user requests **D-VerT** to show the result in the DICE IDE to see if the property is fulfilled or not and, in the negative case, the trace of the system that violates it. Figure 1 shows the main steps of the D-VerT workflow: the creation through the DICE IDE of the UML DTSM model of the application, the automatic transformation from the UML model to the formal model and the actual run verification task.

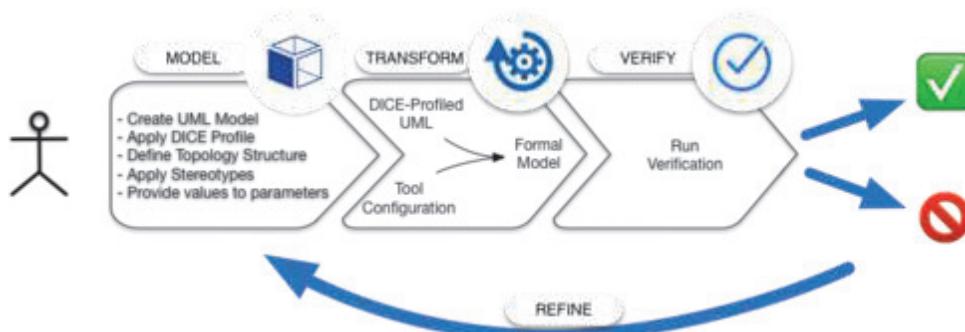


Figure 1. D-VerT workflow

The architecture

As shown in Figure 2, **D-VerT** has a client-server architecture: the client component is an Eclipse plugin that is fully integrated with the DICE IDE and the server component is a RESTful web server. The client component manages the transformation from the DTSM model defined by the user to an intermediate JSON object which is then used to invoke the server. The server component, based on the content of the JSON file generates the formal model which is then fed to the core satisfiability/model checking tool. This tool is in charge of verifying if the property holds for the provided set of formulae, i.e., the formal model of the system.

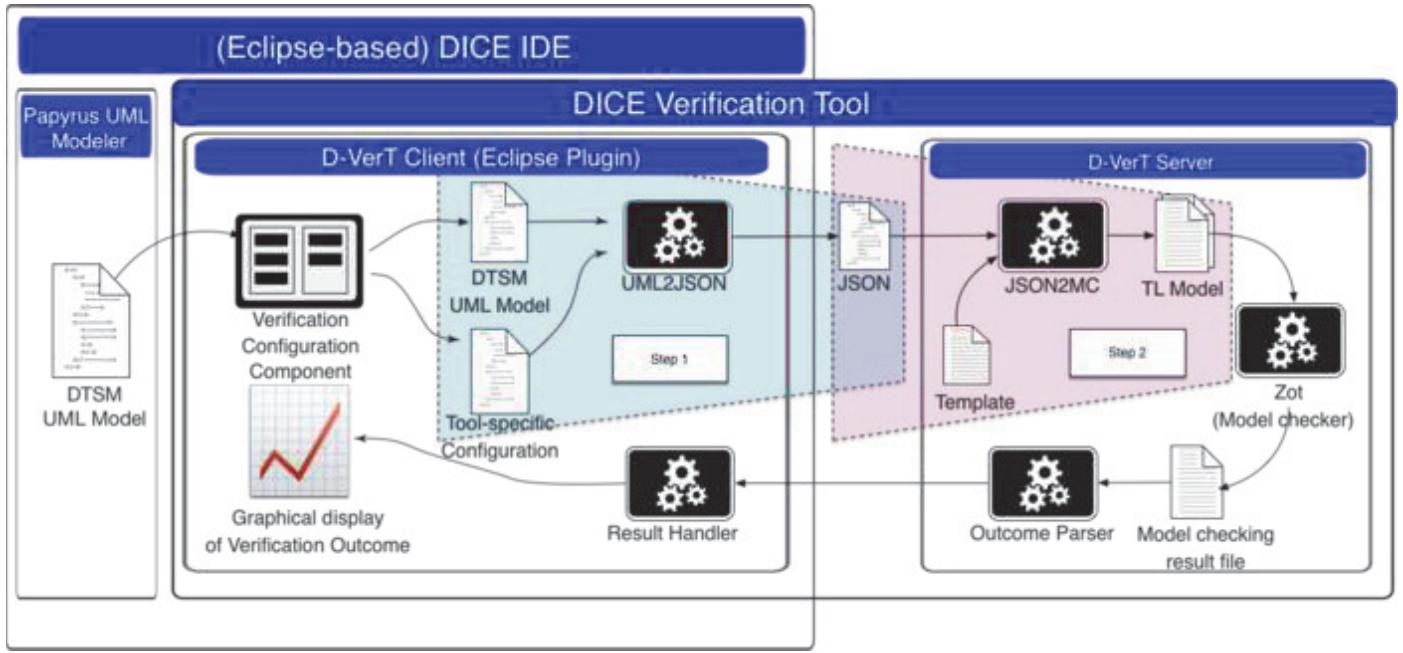


Figure 2. Main components of D-VerT and their interaction.

Open challenges

The usability of the tool is subordinated to the availability of a numerical estimation of some non-functional metrics that are needed by the users of **D-VerT** for annotating DTSM models that undergo verification. These values can be either collected by means of a monitoring platform tracking events from a running application, whose DTSM model is elicited from the implemented solution, or they can be estimated through designers experience or by means of historical data previously collected from other applications that have similarities with the one in analysis.

Application domain

The current version of **D-VerT** supports

- bottleneck analysis for Storm topologies and
- feasibility and boundedness analysis for Spark jobs.

In the case of Storm topologies, the bottleneck analysis helps designers to discover bottleneck nodes as their timing features are not correctly designed. A wrong design of timing constraints might actually cause anomalies such as (i) latency in processing tuples and (ii) monotonic growth of queues occupancy

In the case of the Spark jobs, the feasibility analysis aims at checking if there exist an execution of the system whose duration is lower than a specific deadline, therefore witnessing the feasibility of such execution, while the boundedness analysis checks (making strong assumptions on the idle time of the application) whether all the possible executions of the system are below a certain threshold.

Most of the streaming and batch reference technologies can describe applications by means of graphs representing the computation performed by the application. While, in some cases, such graphs can be derived from the application defined in a declarative way (e.g. in Apache Spark and Apache Flink), other technologies, such as Apache Storm and Apache Samza, allow to directly specify the topology of the application, hence to define applications in a compositional way by combining multiple blocks of computation.

Storm

A Storm application is specified at DTSM level, by means of the topology which implements the application. A topology is a graph which is composed of two classes of nodes.

1. Input nodes (spouts) are sources of information and they do not have any role in the definition of the logic of the application. They can be described through the information related to the stream of data they inject in the topology

- and the features of their computation is not relevant to define the final outcome of the topology
2. Computational nodes (bolts) elaborate input data and produce results which, in turn, are emitted towards other nodes of the topology. The bottleneck analysis focuses on bolts only

In addition, a topology defines exactly the connections among the nodes, being the inter node communication based on message exchange (a message is called *tuple*). Therefore, for any node n , it is statically defined at design time (DTSM model) both the list of nodes subscribing to n - i.e., receiving its emitted messages - and the list of nodes that n is subscribed to.

The following notions (and parameters) are needed by the **D-VerT** user to perform verification.

The behavior of a bolt is defined by a sequence of five different states *idle*, *execute*, *take*, *emit*, *fail* with the following meaning:

- *idle*: no tuples are currently processed in the bolt.
- *execute*: at least one but at most TakeMax tuples are currently elaborated in the bolt.
- *emit*: the bolt emits tuples towards all the bolts that are subscribed.
- *take*: the bolt takes at least one but at most limited number of tuples (defined by the following *parallelism* value) are removed from the queue and initializes a proper number of concurrent thread to process them all.
- *fail*: the bolt is failed and any of the previous states cannot occur

Each bolt has the following parameters:

- σ is a positive real value which abstracts the functionality that is computed by a bolt. If a bolt filters tuples, i.e., the number of incoming tuples is greater than the number of outgoing tuples then $0 < \sigma < 1$, otherwise its value is $\sigma \geq 0$. It expresses a ratio between the size of the input stream and the outgoing stream.
- *parallelism* is the maximum numbers of concurrent threads that may be instantiated in a spout/bolt. Therefore, an active spout/bolt can emit/remove a number of tuples from its queue at the same time which is equal to parallelism value.
- α is a positive real value which represents the amount of time that a bolt requires to elaborate one tuple.
- *rebootTime* (min/max) is a positive real value which represents the amount of time that a bolt requires to restore its functionality after a failure.
- *idleTime* (max) is a positive real value which represents the amount of time that a bolt may be in idle state.
- *timeToFailure* (min) is a positive real value which represents the amount of time between two consequent failures.

Each spout is described by the average emit rate of the tuples that are injected into the topology

Spark

A Spark application is specified at DTSM level, by means of a Directed Acyclic Graph (DAG) of operations transforming data that are aggregated in the nodes, called *stages*.

The fundamental data structure in Spark is the so-called Resilient Distributed Dataset (RDD). RDDs support two types of operations:

- Transformations are operations (such as *map*, *filter*, *join*, *union*, and so on) that are performed on an RDD and which yield a new RDD.
- Actions are operations (such as *reduce*, *count*, *first*, and so on) that return a value obtained by executing a computation on an RDD.

Spark arranges the transformations to maximize the number of operations executed in parallel by scheduling their operations in a proper way. A *stage* is a sequence of transformations that are performed in parallel over many partitions of the data and that are generally concluded by a shuffle operation. Each stage is a computational entity that produces a result as soon as all its constituting operations are completed. Each stage consists of many tasks that carry out the transformations of the stage; a *task* is a unit of computation that is executed on a single partition of data.

The computation realized by a DAG is called *job*. Hence, a DAG defines the functionality of a Spark job by means of an operational workflow that specifies the dependencies among the stages manipulating RDDs. The dependency between two stages is a precedence relation; therefore, a stage can be executed only if all its predecessors have finished their computation.

At runtime, the actual computation is realized through *workers*, *executors* and *tasks* over a cluster of nodes. A *worker* is a node that can run application code in the cluster and that contains some executors, i.e., processes that run tasks and possibly store data on a worker node.

The following notions (and parameters) are needed by theD-VerT user to perform verification.

- *Latency* is an estimation of the duration of the task per data partition, and with
- *Tot_cores* is the number of CPU cores executing the task.
- *Tot_tasks* is the total number of tasks to be executed
- *deadline* is the maximum duration of theSpark application.

The analyses that can be performed are the following:

- feasibility analysis ofSpark jobs verifies whether there exist an execution of the system whose duration is lower than the deadline;
- boundedness analysis checks whether all the possible executions of the system are below the deadline.

A test case - DigitalPebble web crawler

D-VerT is an Eclipse plugin that can be executed within the DICE IDE. To carry out formal verification tasks, Storm topologies and Spark jobs have to be specified by means of UML class diagrams and activity diagrams, respectively, conveniently annotated with the DICE::Storm and DICE::Spark profiles. The diagrams can easily be drawn by dragging and dropping Eclipse items that are available in the DICE toolbars whereas the annotations can be specified in the bottom panel of the DICE IDE.

Figure 3 depicts the Storm topology implementing the DigitalPebble web crawler. Each node, either spout or bolt, has been annotated with the parameter values that are required to perform the bottleneck analysis.

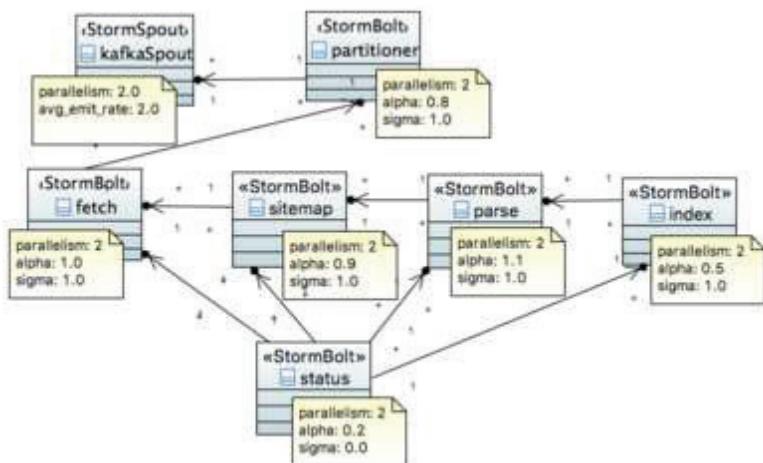


Figure 3. Digital Pebble Topology

D-VerT can be set through a Run Configuration window allowing users to launch the analysis. The parameters that the user can specify are the following:

- the file containing the model undergoing the analysis,
- the time bound limiting the length of the executions that the solver considers in the analysis of the topologies. Being D-VerT based on boundedmodel checking approach, the time bound is the maximum number of distinct and consecutive events that the solver can use to produce an execution of the topology
- The solver plugin to be used in the analysis,
- The set of monitored bolts that are analyzed, and
- The IP address and the port where theD-VerT server is running.

The analysis allows for the detection of possible runs of the system leading to an unbounded growth of at least one of the bolts' queues. If D-VerT detects a problem, it returns a trace witnessing the existence of such a run of the system—i.e., the counterexample violating the boundedness property. The trace is returned to the user both in a textual format (i.e., the bare output of underlying solver) and in a graphical format. Figure 4 shows an example of such output trace. It represents the evolution of the number of tuples in a bolt's queue over time. The trace is composed by a prefix and a suffix: the latter, highlighted by the gray background, captures the growing trend of the queue size, as it corresponds to a series of operations in the system that can be repeated infinitely many times. When no trace is detected, the result is UNSA.

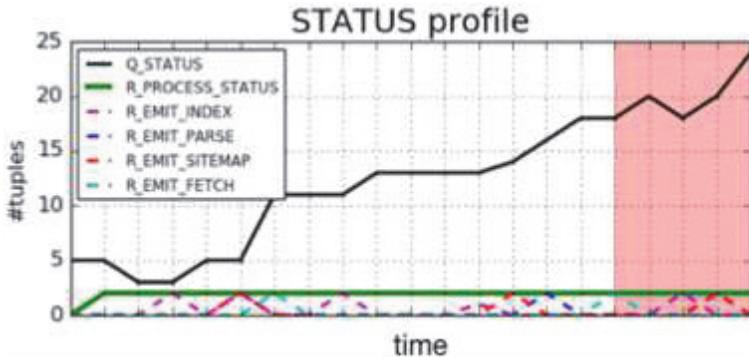


Figure 4. D-VerT output trace

Figure 5 shows a Spark DAG implementing the WordCount application.

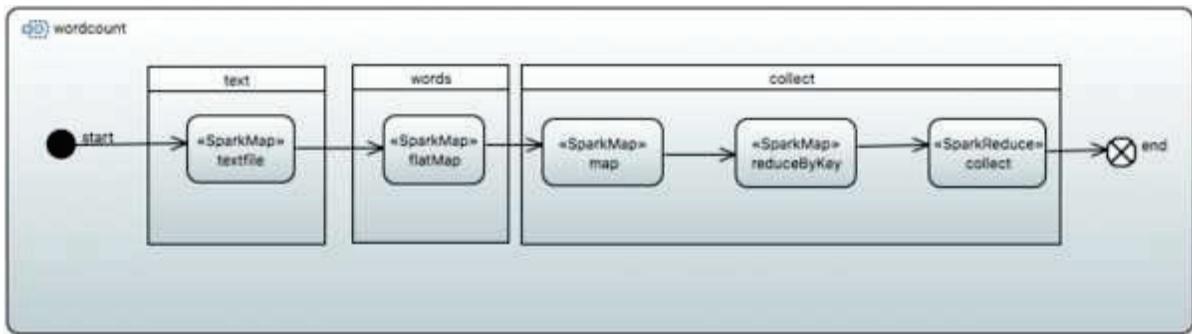


Figure 5. WordCount DAG

The Run Configuration dialog, which **D-VerT** shows to users, allows them to choose the kind of analysis to perform, either feasibility or boundedness, beside the parameters defining the model to be used and the server IP and port.

Spark analysis returns a boolean outcome (yes/no) which specifies whether the property is violated or not. The negative response is supplied with a counterexample proving the violation.

Conclusion

The main achievement of verification in DICE has been the development of abstract models to support safety verification of Apache Storm and Spark applications. The models have been built by means of a logical approach. They were designed to be configurable and were provided with a layered structure, which facilitates the integration with the DICE framework by decoupling the verification layer from the modeling DTSM diagrams. Ad-hoc model-to-model transformations were developed to obtain verifiable models from the user design, the latter realized through the verification toolkit in the DICE IDE.

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Quality_Verification&oldid=3367302'

This page was last edited on 26 January 2018, at 17:37.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Quality Simulation

Contents

Introduction

Motivation

- Performance Metrics
- Reliability Metrics

Existing Solutions

How the tool works

Open Challenges

Application domain: known uses

Conclusion

References

Introduction

Quality assurance of DIA that use Big Data technologies is still an open issue. We have defined a quality-driven framework for developing DIA based on MDE techniques. Here, we propose the architecture of a tool for predicting quality of DIA. In particular, the quality dimensions we are interested are efficiency and reliability. This tool architecture addresses the simulation of the behaviour of a DIA using Petri net models.

In our view software non-functional properties follow the definition of the ISO^[1]/IEC^[2] standards and may be summarised as follows:

- Reliability: The capability of a software product to maintain a specified level of performance, including Availability and Fault tolerance.
- Performance: The capability of a software product to provide appropriate performance, relative to the amount of resources used, as described by time behaviour and resource utilisation. The terms performance and efficiency are considered synonyms throughout.

This chapter introduces definitions of reliability and performance properties and metrics that are relevant to the analysis. Service-Level Agreements (SLAs) are readily determined by predicated on these quantities. SLAs can be directly annotated in the UML models, thus we do not look at other forms of specification (e.g. Web Services (WS) standards^[3]).

Motivation

This section reviews some basic definitions concerning performance prediction metrics. These are standard definitions that are usually used in the context of queueing models, but are also applicable to performance predictions obtained with other formalisms, such as Stochastic Petri Nets^[4]. We give definitions for the basic case where requests are considered of a single type (i.e. a single class model). The generalisation for multiple types is simple and in most cases requires only adding an index to each metric to indicate the class of requests it refers to^[5].

Performance Metrics

Considering an abstract system where T is the length of time we observe a system, A is the number of arrivals into the system, and C is the number of completions (users leaving the system) we can define the commonly used measures^[5]:

- Arrival rate: A / T
- Throughput: $X = C / T$ (simply the rate of request completions)

In a system with a single resource we can measure B_k as the time the resource k was observed to be busy, and denote C_k the number of arrivals at the resource. If T_k is the length of the observation time, we can then define two additional measures:

- Utilisation, $U_k = B_k / T_k$
- Service time per request, $S_k = B_k / C_k$

From these measures we can derive the Utilisation Law as $U = XS$. The above quantities can be made specific to a given resource, for example U_k stands for the utilisation of resource k. One of the most useful fundamental laws is Little's Law^[5] which states that N, the average number of customers in a system, is equal to the product of X, the throughput of the system, and R, the average response time a customer stays in the system. Formally this gives: $N = XR$. The formula is unchanged if one considers a closed model, which has a fixed population N of customers. Another fundamental law of queueing systems is the Forced Flow Law^[5] which, informally, states that the throughputs in all parts of the system must be proportional to one another. Formally the Forced Flow Law is given by: $X_k = V_k$

X_k , where X_k is the throughput at resource k , V_k is the visit count of resource k , i.e., the mean number of times users hit this resource. Combining both Little's Law and the Forced Flow Law allows for a wide range of scenarios to be analysed and solved for a particular desired quantity. For example, it is possible to compute utilisation at a server k in a distributed network directly as^[5] $U_k = X_k D_k$ where $D_k = V_k S_k$ is called the service demand at server k .

Reliability Metrics

The area of reliability prediction is established and focuses on determining the value for a number of standard metrics, which we review below. The execution time or calendar time is appropriate to define the reliability as $R(t) = \text{Prob}\{\mathbf{T} > t\}$ that is, reliability at time t is the probability that the time to failure \mathbf{T} is greater than t or, the probability that the system is functioning correctly during the time interval $(0, t]$. Considering that $F(t) = 1 - R(t)$ (i.e., unreliability) is a probability distribution function, we can calculate the expectation of the random variable \mathbf{T} as $\int_0^\infty t dF(t) = \int_0^\infty R(t) dt$. This is called Mean Time to Failure (MTTF)^[6] and represents the expected time until the next failure will be observed.

The failure rate (called also rate of occurrence of failures) represents the probability that a component fails between $(t, t+dt)$, assuming that it has survived until the instant t , and is defined as a function of $R(t)$: $h(t) = -\frac{1}{R(t)} \frac{dR(t)}{dt}$. The cumulative failure function denotes the average cumulative failures associated with each point in time, $E[N(t)]$.

Maintainability is measured by the probability that the time to repair (\mathbf{O}) falls into the interval $(0, t]$ ^[6] $M(t) = \text{Prob}\{ \mathbf{O} \leq t \}$. Similarly, we can calculate the expectation of the random variable \mathbf{O} as $\int_0^\infty t dM(t)$, that is called MTTR (Mean Time To Repair), and the repair rate as $\frac{dM(t)}{dt} \frac{1}{1 - M(t)}$.

A key reliability measure for systems that can be repaired or restored is the MTBF (Mean Time Between Failures)^[6], that is the expected time between two successive failures of a system. The system/service reliability on-demand is the probability of success of the service when requested. When the average time to complete a service is known, then it might be possible to convert between MTBF and reliability on-demand.

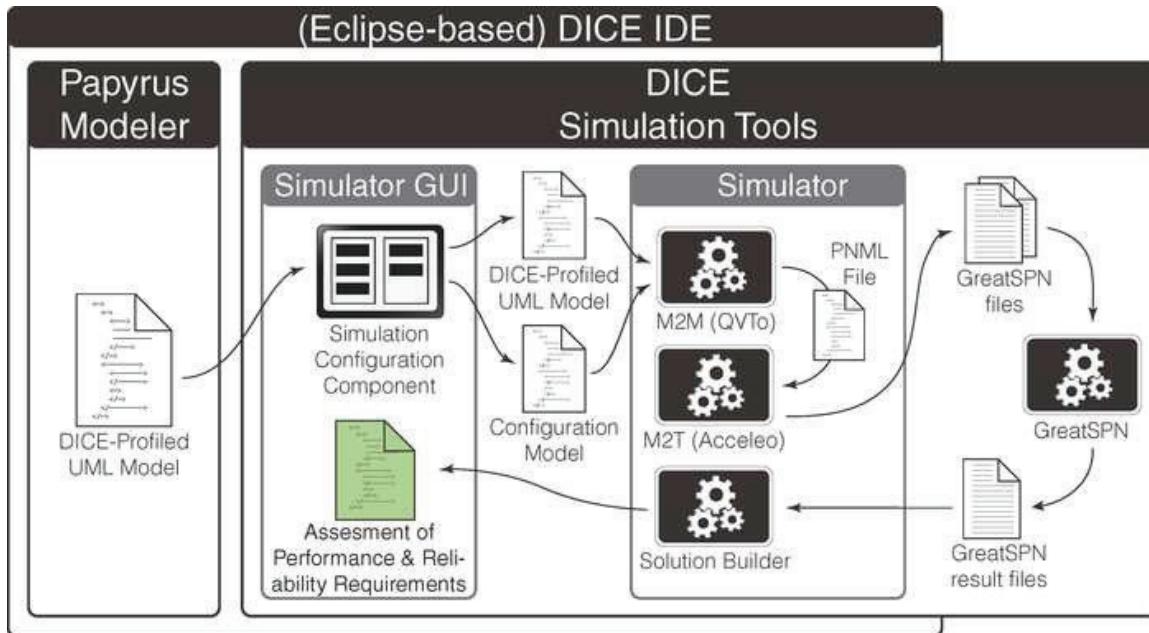
Availability is defined as the probability that the system is functioning correctly at a given instant $A(t) = \text{Prob}\{\text{state=UP} | \text{time}=t\}$. In particular, the steady state availability can be expressed as function of MTTF and MTTR (or MTBF): $\text{Availability}_\infty = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$.

Existing Solutions

While there exist multiple tools that can simulate software models and obtain its quality properties, there is not any tool that offers the capabilities to simulate, from software design models, the quality of applications that use the Big Data technologies considered in DICE.

How the tool works

Next image shows a possible architecture of a Simulation tool and the internal data flows.



Architecture of a Simulation Tool

Next, we provide a description of the different modules, the data they share, and their nature:

1. The DICE-IDE is an Eclipse-based environment in which the different components are integrated.
2. A simulation process starts by defining a set of DICE-Profiled UML models. For this stage, a pre-existing modeling tool is used. Papyrus UML is one of the open source UML modelling tools that support the MAIRE, in which the DICE profile is based on. As proposed in the technical

- Report [7], this component/tool is used to perform the initial modelling stage.
3. When the user (the QA Engineer) wants to simulate a model, he/she uses the Simulator GUI to start a simulation. The Simulator GUI is an ad hoc Eclipse component that contributes a set of graphical interfaces to the DICE-IDE. These interfaces are tightly integrated within the DICE-IDE providing a transparent way for interacting with the underlying analysis tools. The Simulation Configuration Component is a sub-componer of the Simulator GUI. It is in charge of: (i) asking for the model to be simulated (using the DICE-IDE infrastructure, dialogs, etc.); and (ii) asking for any additional data required by the Simulator
 4. When the user has finished the configuration of a simulation, the Configuration tool passes two different files to the Simulator: the DICE-profiled UML model (i.e., the model to be analysed) and the Configuration model. The Simulator is an ad hoc OSGi component that runs in background. It has been specifically designed to orchestrate the interaction among the different tools that perform the actual analysis.
 5. The Simulator executes the following steps: (i) transforms the UML model into a PNML file using a M2M transformation tool; (ii) converts the previous PNML file to a GreatSPN-readable file using a M2T transformation tool; (iii) evaluates the GreatSPN-readable file using the GreatSPN tool; and (iv) builds a tool-independent solution from the tool-specific file produced by GreatSPN. To execute the M2M transformations we have selected the eclipse QVT transformations engine. QVT [8] is the standard language proposed by the OMG (the same organism behind the UML and MARTE standards) to define M2M transformations. QVT proposes three possible languages to define model transformations: operational mappings (QVTO, imperative, low-level), core (QVTC, declarative, low-level) and relations (QVTR, declarative, high-level). However, although there are important efforts to provide implementations for all of them, only the one for QVTR is production-ready and as such is the chosen one. To execute the M2T transformations we have selected Acceleo [9]. Starting from Acceleo 3, the language used to define an Acceleo transformation is an implementation of the MOFM2T standard [10], proposed by the OMG too. In this sense, we have selected Acceleo to make all our toolchain compliant to the OMG standards, from the definition of the initial (profiled) UML models to the 3rd party analysis tools (which use a proprietary format). The analysis is performed using the GreatSPN tool. GreatSPN is a complete framework for the modeling, analysis and simulation of Petri nets. This tool can leverage those classes of Petri nets needed by our simulation framework, i.e., Generalized Stochastic Petri Nets (GSPN) and their colored version, namely Stochastic Well-formed Nets (SWN). GreatSPN includes a wide range of GSPN/SWN solvers for the computation of performance and reliability metrics (the reader can refer to the "State of the art analysis" deliverable D1.1 for details about the GreatSPN functionalities).
 6. Finally, the tool-independent report produced by the Simulator is presented in the DICE-IDE using a graphical component of the Simulator GUI. This component provides a comprehensive Assessment of Performance and Reliability Metrics report in terms of the concepts defined in the initial UML model.

Open Challenges

The open challenges in the simulation of software applications that use Big Data technologies are shared with the general simulation of software systems. Next list describes three of the main challenges:

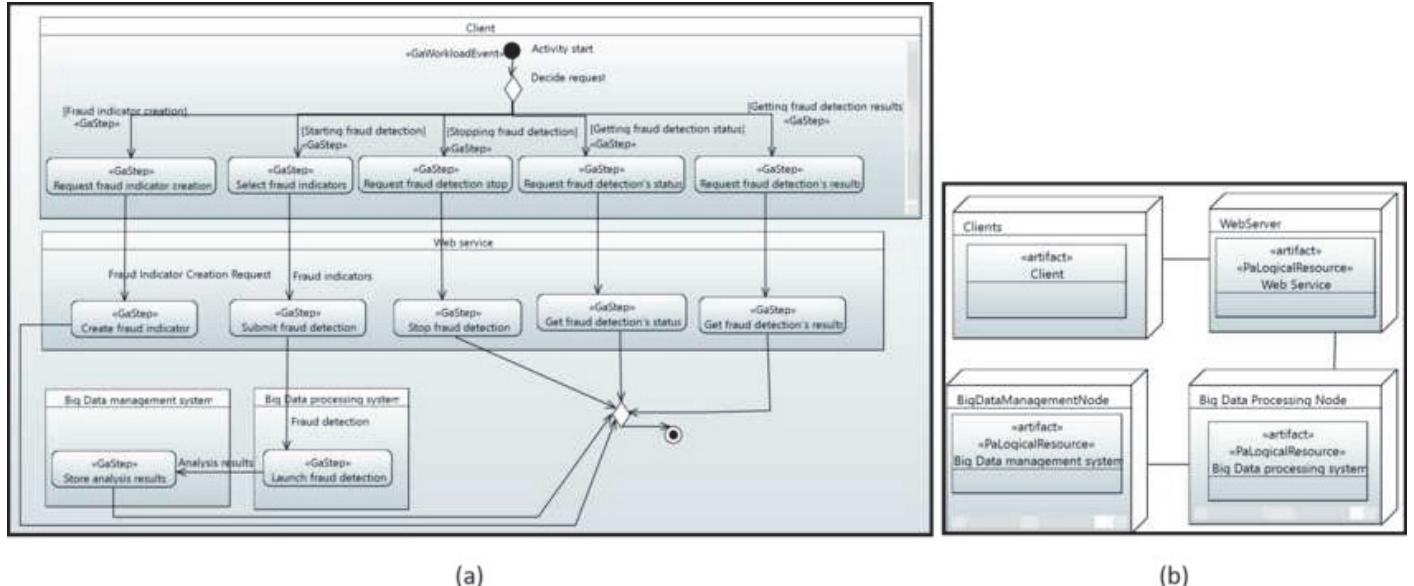
- **Obtain accurate model parameters** Some information that users shall provide in the UML models is not easy to obtain, such as the execution time of activities or probabilities of execution of each branch in operators of condition. For instance, users would need powerful monitors to measure the exact values of model parameters, or process mining techniques acting upon logs of the running application to discover them, or significant expertise to estimate them. Although the Simulation tool implements what-if analysis to relieve users of knowing the exact value of some parameters, the creation and evaluation of a single exact/accurate model is still an open challenge
- **Usability in model generation** the generation of profiled UML models that are the input of the simulation tool is done through Papyrus tool. These profiled models use the DICE profiles, which are in turn based on the standard MARTE profile. For users who are non-experts on the utilization of MARTE, the meaning, purpose and definition of some attributes that are inherited by DICE from MARTE stereotypes may not be clear from the very beginning. This utilization of standard profiles might reduce the slope of the learning curve for creating correct inputs for the tool.
- **Simulation in presence of rare events** the presence of rare events hinders the evaluation of systems based on discrete event simulation. For instance, this may happen in conditional branches whose execution probability is very low. In this case, the current implementation of the tool would require much more simulation time to generate results with high confidence. Research advances have been achieved in this challenge, and techniques for simulation in presence of rare have been proposed. Equipping the Simulation tool with some of these techniques is at present an open challenge and part of future work.

Application domain: known uses

The Quality Simulation through the implemented Simulation Tool has been applied to the BigBlu application. BigBlu is an e-government software system developed for Tax Fraud Detection which manages large quantity of data from taxpayers. A more detailed description of BigBlu is provided in [Fraud Detection](#)

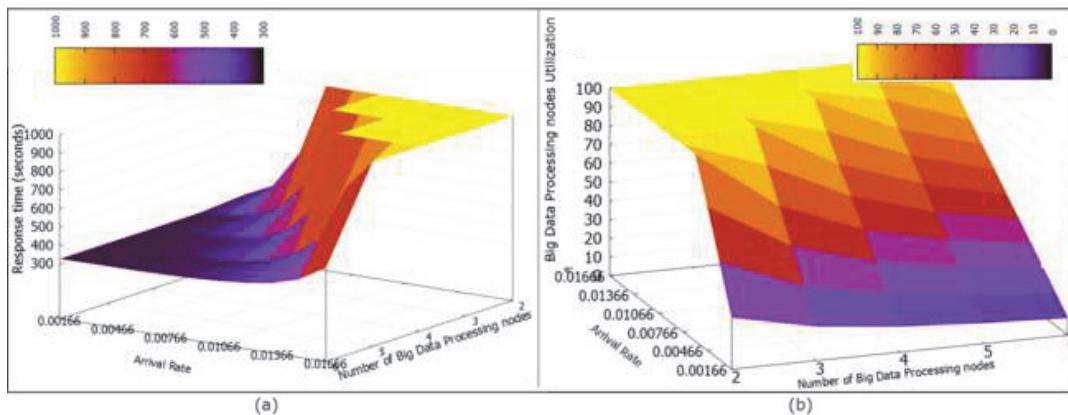
Next image depicts the UML activity and deployment diagrams created with Papyrus tool that are the inputs for the Simulation Tool. These models are annotated with DICE [11] and MARTE profiles. More in detail, part (a) shows the workflow of the application. It begins with an initial node, followed by a decision node which divides the execution workflow in multiple paths, depending on the decision condition (e.g., "Fraud indicator creation" in the first branch). Each path has several activities that represent the particular execution steps. These activity nodes are stereotyped as <>GaStep>>, a stereotype from MARTE. <>GaStep>> allows capturing performance properties of the application operation, such as the expected execution time of the task or the probability to be carried out. Finally, every path converges to a merge node and then the workflow finishes.

Every partition of the activity diagram in part (a) is mapped to an artefact, which is hosted by a device in part (b). Therefore diagram (b) represents the deployment of the application in physical devices, which can also be refined by MARTE stereotypes (e.g., PaLogicalResource) for capturing the hardware details.



BigBlu design models provided to the Simulation Tool

The Simulation Tool has been applied to evaluate the expected performance of BigBlu --in terms of response time of execution requests and utilization of resources-- with respect to both the intensity with which the application is used (i.e., the arrival rate of requests) and the number of *Big Data Processing Nodes* deployed (see that *Big Data Processing Nodes* are the computing nodes that execute the *Launch fraud detection* activity). The type of results obtained are depicted in next figure, whose part (a) depicts the expected response time of BigBlu varying the arrival rate of requests and number of processing nodes and part (b) depicts the expected utilization of resources. A more detailed study input values in models and experimentation can be found in [12].



Examples of Simulation tool performance results

Conclusion

This chapter has motivated the simulation of software applications that use Big Data technologies for evaluating their quality in terms of performance and reliability. It has also presented the DICE Simulation Tool, a software tool that implements this quality simulation. The DICE Simulation Tool is able to cover all the steps of a simulation workflow: from the design of the model to simulate, its model transformation to analysable models, the simulation of the analysable model to compute its properties, to the retrieval of the quality results to the user in the domain of the design model through a userfriendly GUI.

References

1. "ISO Standards". <https://www.iso.org/standards.html>
2. "IEC Standards". <http://www.iec.ch/>
3. "List of Web Services standards". https://en.wikipedia.org/wiki/List_of_web_service_specifications#Web_Service_Standards_Listings
4. Molloy, M.K. (1981). *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. PhD thesis, UCLA, Los Angeles (CA).
5. Lazowska, E.D.; Zahorjan, J.; Sevcik, K.C. (1984) *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall Inc. ISBN 0-13-746975-6
6. Johnson, B.W. (1989). *Design and Analysis of Fault-Tolerant Digital Systems*. Addison-Wesley. ISBN 0-201-07570-9.
7. The DICE Consortium (2015). *State of the Art Analysis(Report)*. http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2015/08/D1.1_State-of-the-art-analysis1.pdf
8. Object Management Group (OMG) (2011) *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification 1.1*. <http://www.omg.org/spec/QVT/1.1/>

9. The Eclipse Foundation & Obeo."Acceleo". <https://eclipse.org/acceleo/> Retrieved 1 December 2015
 10. Object Management Group (OMG) (2008)*MOF Model to Text Transformation Language (MOFM2T)* 1.0. <http://www.omg.org/spec/MOFM2T/1.0/>
 11. A.Gómez; J.Mersegue E. Di Nitto, D.A. Tamburri (2016). "Towards a UML profile for data intensive applications". 2rd International Workshop on Quality-Aware DevOps (QUDOS). ACM. pp. 18-23. doi:<https://doi.org/10.1145/2945408.2945412> ISBN 978-1-4503-4411-1 <https://zaguan.unizares/record/60722?ln=es>
 12. D. Perez-Palacin; YRidene, J.Mersegue (2017)."Quality Assessment in DevOps: Automated Analysis of a Fraud Detection System" 3rd International Workshop on Quality-Aware DevOps (QUDOS). ACM. pp. 133-138doi:<https://doi.org/10.1145/3053600.3053632> ISBN 978-1-4503-4899-7. <https://zaguan.unizares/record/61309?ln=es>
-

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Quality_Simulation&oldid=3364863'

This page was last edited on 22 January 2018, at 14:27.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Quality Optimisation

Contents

Introduction
Motivations
Existing solutions
How the tool works
 The optimization process
 The Architecture
Open Challenges
Application Domain
 A test case – NETF Fraud detection application
Experimental Results
Conclusion

Introduction

As discussed in [Section 1](#), the last years have seen a steep rise in data generation worldwide, with the development and widespread adoption of several software projects targeting the Big Data paradigm. Many companies currently engage in Big Data analytics as part of their core business activities, nonetheless, there are no tools and techniques to support the design of the underlying hardware configuration backing such systems. The DICE optimisation tool (codename D-SPACE4Cloud) is a software tool that supports this task. It is able to support software architects and system operators in the capacity planning process of shared Hadoop Cloud.

Motivations

Nowadays, data intensive application adoption has moved from experimental projects to mission-critical, enterprise-wide deployments providing a competitive advantage and business innovation. The deployment and setup of a big data cluster are time-consuming activities. Initially, MapReduce jobs were meant to run on dedicated clusters. Now, data intensive applications have evolved and large queries, submitted by different users, need to be performed on shared clusters, possibly with some guarantees on their duration. Capacity allocation becomes one of the most important aspects. Determining the optimal number of nodes in a cluster shared among multiple users performing heterogeneous tasks is an important and difficult problem.

Existing solutions

To the best of authors' knowledge, there are no available tools offering such peculiarities that are specifically developed for the DICE reference technologies.

How the tool works

D-SPACE4Cloud is a novel tool implementing a battery of optimization and prediction techniques integrated so as to efficiently assess several alternative resource configurations, in order to determine the minimum cost cluster deployment satisfying Quality of Service (QoS) constraints. In a nutshell, the tool implements a search space exploration able to determine the optimal virtual machine (VM) type and the number of instance replicas for a set of concurrent applications. The underlying optimization problem is demonstrated to be NP-hard and it is solved heuristically whereas applications execution time or throughput are estimated via queueing network (QN) or Stochastic Work Formed Net (SWN) models.

The optimization process

Two main analyses can be conducted using D-SPACE4Cloud:

1. Public Cloud Analysis: In this scenario, the software architect or the system operator wants to analyse the case in which the whole big data cluster is provisioned on public cloud. The first consequence of this choice is that the virtualized resources (i.e., VMs) that can be used to provision the cluster can be considered practically infinite. This also means that, under the common assumption that rejecting a job has a much higher cost than the VM leasing costs, it will not apply any job rejection policy in this case. Consequently the concurrency level for each job, i.e., the number of concurrent user running the data intensive application can be set arbitrarily high being always (theoretically) possible to provision a cluster able to handle the load. In this scenario, a system operator may want to know which machine type to select and how many of them in order to execute the application with a certain level of concurrency, meaning considering several similar applications running at the same time in the cluster. She/he might also like to know which cloud provider is cheaper to choose, considering that providers have also different pricing models.
2. Private Cloud Analysis: In this case the cluster is provisioned in house. This choice in some sense changes radically the problem. In fact, the resources used to constitute a cluster are generally limited by the hardware available. Therefore, the resource provisioning problem has to contemplate the possibility to exhaust the computational capacity (memory and CPUs) before being able to provision a cluster capable of satisfying a certain concurrency level and deadlines. In such a situation, the software architect or the system operator could consider two sub-scenarios:
 1. Allowing job rejection, that is consider the possibility to reject a certain number of jobs (lowering consequently the concurrency level), i.e., introducing an Admission Control (AC) mechanism. In this case, since the overall capacity is limited, the system reacts to the excess of load by rejecting jobs; this has an impact on the execution costs as it seems fair to believe that pecuniary penalties can be associated to rejection.
 2. Denying job rejection, that is imposing that a certain concurrency level must be respected. This translates into a strong constraint for the problem that may not be satisfiable with the resources at hand.

In this scenario, D-SPACE4Cloud estimates the total capacity of an in-house cluster required to support data intensive applications execution minimising the Total Cost of ownership (TCO) of the system, which includes physical servers acquisition costs, electricity costs and, possibly, penalties for job rejections.

The Architecture

D-SPACE4Cloud is a distributed software system able to exploit multi-core architecture to execute the optimization in parallel, which encompasses different modules that communicate by means of RESTful interfaces or SSH following the Service Oriented Architecture (SOA) paradigm. In particular, it features a presentation layer (an Eclipse plug-in), an orchestration service (referred to as frontend) and a horizontally scalable optimization service (referred to as backend), which makes use of third-party services as RDBMS, simulators and mathematical solvers. The tool implements an optimization mechanism that efficiently explores the space of possible configurations, henceforth referred to as Solution space. Figure 1 depicts the main elements of the D-SPACE4Cloud architecture that come into play in the optimization scenario. The Eclipse plug-in allows the software architect to specify the input models and performance constraints and transforms the input DICE UML diagrams into the input performance models for the performance solver (GreatSPN or JMT). The frontend exposes a graphical interface designed to facilitate the download of the optimization results (which are computed through batch jobs) while the backend implements a strategy aimed at identifying the minimum cost deployment. Multiple DTSMs are provided as input, one for each VMs considered as a candidate deployment. VMs can be associated with different cloud providers. D-SPACE4Cloud will identify the VM type and the corresponding number which fulfill performance constraints and minimize costs. The tool takes as input also a DDSM model, which is updated with the final solution found and can be automatically deployed through the DICER tool. Moreover, the tool requires as input a description of the execution environment (list of providers, list of VM types or a description of the computational power available in house) and the performance constraints. Input files and parameters can be specified by the user through a wizard. D-SPACE4Cloud exploits the model-to-model transformation mechanism implemented within the DICE Simulation tool to generate a suitable performance model, i.e., SWN or QN to be used to predict the expected execution time for Hadoop MapReduce or Spark DIAs or cluster utilization for Storm. The Initial Solution Builder generates a starting solution for the problem using a Mixed Integer Non-linear Programming (MINLP) formulation where the job duration is expressed by means of a convex function: Figure 1 provides further details. The fast MINLP model is exploited to determine the most cost effective VM type for all applications. Yet the quality of the returned solution can still be improved since the MINLP problem is just an approximate model. For this reason, a more precise QN or SWN is adopted to get a more accurate execution time assessment for each application: the increased accuracy leaves room for further cost reduction. However, since QN or SWN simulations are time-consuming, the space of possible cluster configurations has to be explored in the most efficient way, avoiding evaluating unpromising configurations.

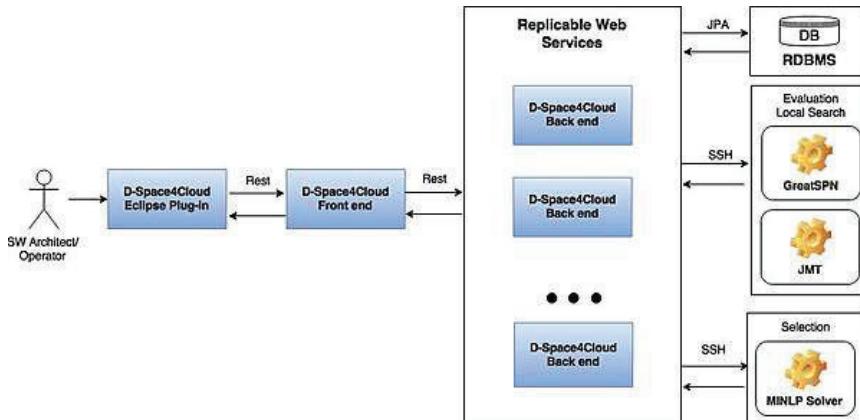


Figure 1. D-SPACE4Cloud architecture

In the light of such considerations, a heuristic approach has been adopted and a component called Parallel LS Optimizer has been devised. Internally, it implements a parallel Hill Climbing (HC) technique to optimize the number of replicas of the assigned resource for each application; the goal is to find the minimum number of resources to fulfil the QoS requirements. This procedure is applied independently, and in parallel, on all application classes and terminates when a further reduction in the number of replicas would lead to an infeasible solution. In particular, HC is a local-search-based procedure that operates on the current solution performing change (more often referred to as move) in the structure of the solution in such a way that the newly generated solution could possibly show an improved objective value. If the move is successful it is applied again on the new solution and the process is repeated until no further improvement is possible. The HC algorithm stops when a local optimum is found; however, if the objective to optimize is convex, HC is able to find the global optimum for the problem. This is the case of the considered cost function, which depends linearly on the number of VMs in the cluster since the VM types to use is fixed in the first phase of the optimization process based on MINLP techniques. In other words, the joint selection of the VM type and their number is NP-hard, but when the type of VM is fixed in the first phase, the HC obtains the final solution for all classes in polynomial time. The HC move consists in increasing/decreasing and changing the VM type for each DIA. This task is executed in parallel where possible, considering SLAs and available resources (in case of private cloud) as constraints. Parallelism is particularly important as each solution spawned during the optimization is evaluated via simulation and this is a time-consuming task. Therefore, in order to make the DICE optimization tool usable, we focused on increasing the level of parallelism as much as possible. The final minimum cost configuration found is then returned to the D-SPACE4Cloud Eclipse plug-in and is translated into the DDSM which is ready for the TOSCA Model to Text transformation and deployment implemented by the DICER tool.

Open Challenges

Even if D-Space4Cloud implements an efficient parallel search, simulation is still time-consuming. The development of an ad-hoc simulator which significantly speed-ups the design space exploration time is on-going. In this way we expect that optimization time can be reduced from hours (in the current implementation) to minutes. Moreover, modelling complex application DAGs requires also a significant effort. The development of a log processor, which automatically builds the required DICE UML models from application logs (once early DIA code implementation is available) to simplify such process is also on-going.

Application Domain

D-SPACE4Cloud supports the capacity planning process of shared Hadoop Cloud clusters running MapReduce or Spark applications with deadline guarantees or Storm topologies with guarantees on cluster utilization.

A test case – NETF Fraud detection application

D-SPACE4Cloud has been used by Netflective Technology with the aim to evaluate the cost impact of the implementation of some privacy mechanisms on the taxpayers' data. As discussed in Section 20, the Cassandra databases are filled with taxpayers' details, historical tax declarations, tax payments, and so on.

To achieve anonymization, the first privacy mechanisms applied is masking (see, e.g., Vieria, M.; Madeira, H. "Towards a Security Benchmark for database Management Systems.", in DSN 2015 Proceedings). For that purpose, a dictionary table has been introduced, which implements a one-to-one mapping between a clear ID and a masked ID. In other words, the Cassandra tables store masked IDs for the taxpayers, while the clear ID is available only in the dictionary table which has a restricted access. As an example, Figure 2 shows the three

reference queries we considered during the DICE project, i.e., Query 1, 5 and 7 and an example of an anonymised query. Query 1 accesses two tables to perform its analysis: Declare and TaxPayer. It intends to measure the difference between incomes earned by a taxpayer during two successive years. This is carried out to detect fraudsters by comparing the two incomes according to some criteria. For instance, if the income received a certain year is less than 20% (this percentage can be set as a parameter) than the one received the previous year, then the taxpayer is suspect. Since incomes are stored in the table Declare, Query 1 executes two joins: the first to make sure that the two tax declarations relate to the same taxpayer; and the second to obtain the full set of information about her/him. The result of this query is saved and used by other queries by passing the expected arguments, such as the percentage of income decrease, and the number of years to be taken into consideration. Query 5 involves only the table Declare. This table contains all the information needed to know every individual income and other credentials helpful to justify the amount of tax to be paid. Query 7 involves three tables: TaxPayer, TaxDeclaration, and Signatory. Each tax return must be signed by the taxpayers before they submit it to the fiscal agency.

Query 3 is derived from Query 1 and adds a join to read the clear IDs. Similarly, Query 6 and Query 8 are derived from Query 5 and Query 7 respectively (but are omitted here for simplicity). The second privacy mechanisms considered is encryption with AES at 128 and 256 bit. In this case, the IDs and sensitive data stored in the Cassandra tables are encrypted and decryption is performed contextually while running, e.g., Query 1.

Figure 2. NETF-Case study reference queries

<p>Query 1:</p> <pre>SELECT tp.id, , tp.gender tp.birthdate, tp.birthdepartment, tp.birthcommune, d1.taxdeclaration, d1.declarationdate, d1.income, d2.taxdeclaration AS D2TAXDECLARATION, d2.declarationdate AS D2DECLARATIONDATE, d2.income AS D2INCOME FROM Declare d1 INNER JOIN Declare d2 ON d1.taxpayer = d2.taxpayer INNER JOIN taxpayer tp ON d1.taxpayer = tp.id</pre>
<p>Query 5:</p> <pre>SELECT * FROM Declare d1</pre>
<p>Query 7:</p> <pre>SELECT tp.id,s.location, td.roomcount FROM taxdeclaration td, signatory s, taxpayer tp WHERE s.taxpayer = tp.id AND s.taxdeclaration = td.id</pre>
<p>Query 3:</p> <pre>SELECT dic.und_id, , tp.gender tp.birthdate, tp.birthdepartment, tp.birthcommune, d1.taxdeclaration,d1.declarationdate, d1.income, d2.taxdeclaration AS D2TAXDECLARATION,d2.declarationdate AS D2DECLARATIONDATE, d2.income AS D2INCOME FROM Declare d1 INNER JOIN Declare d2 ON d1.taxpayer = d2.taxpayer INNER JOIN taxpayer tp ON d1.taxpayer = tp.id INNER JOIN dictionary dic ON dic.id=tp.id</pre>

The queries reported in Figure 2 have been implemented in Spark 2.0 and ran on Microsoft Azure HDInsight on D12v2 VMs with 8 cores and 28GB of memory. The number of executors per node (virtual machine) was either two or four. We set the maximum executor memory 8 GB as for the driver memory while executors were assigned two or four cores. The number of nodes varied between 3 and 12. Overall we ran experiments on up to 48 cores. Runs were performed considering the default HDInsight configuration. This performance campaign has been performed to identify the queries profiles. In the following, the baseline Query 5 and its anonymised versions will be compared considering the same configuration.

Here, we present our approach to model Spark applications for performance evaluation using UML diagrams. The Spark application is composed of transformations and actions. D-SPACE4Cloud interprets the UML activity diagram as the DAG of the Spark application, i.e., an activity diagram represents the execution workflow over the application RDDs. A Spark application manages one or more RDDs and therefore, our UML activity diagram accepts several initial and final nodes. Each stage shows the operations executed for each RDD.

Figure 3 shows the Activity Diagram for Query 8 consists of 8 transformations. The UML fork and join nodes are also supported following standard UML modelling semantics. The activity nodes (actions, forks and joins) in the UML activity diagram are grouped by UML partitions (e.g., Transformation and Action).

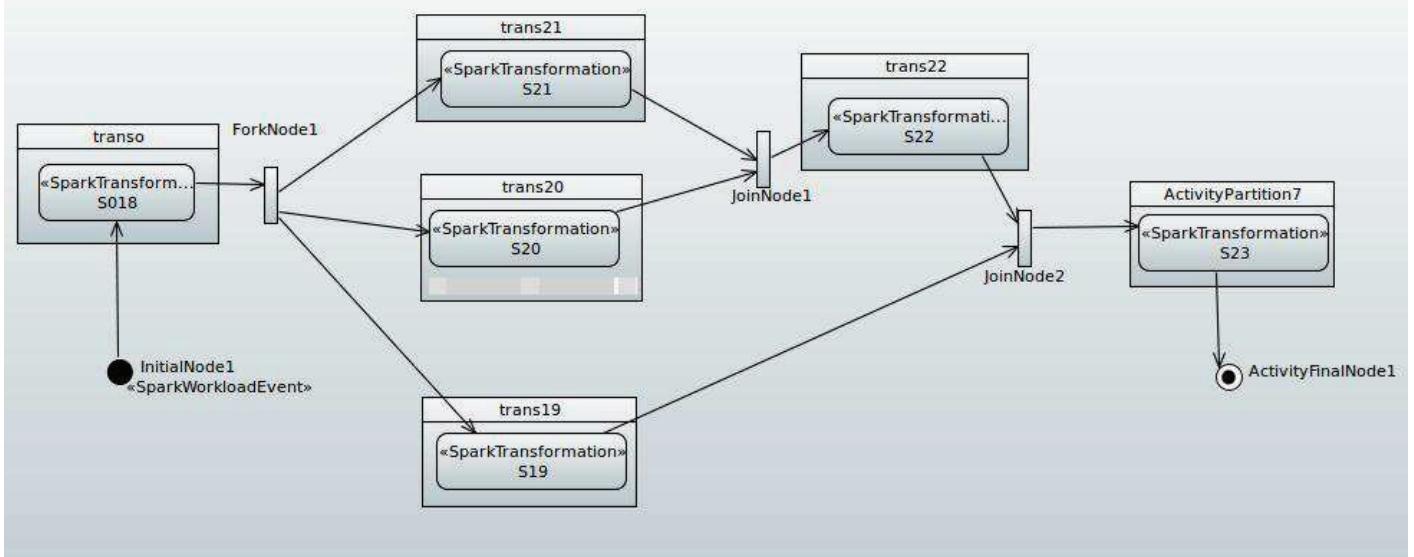


Figure 3. Query8-Activity diagram

A UML profile needs to be applied on each activity and partition nodes in our UML representation. A UML profile is a set of stereotypes that can be applied to UML model elements for extending their semantics.

The Spark profile heavily relies on the standard MARTE profile. This is because MARTE offers the GQAM sub-profile, a complete framework for quantitative analysis, which is indeed specialized for performance analysis, then perfectly matching to our purposes. Moreover, MARTE offers the NFPs and VSL sub-profiles. The NFP sub-profile aims to describe the non-functional properties of a system, performance in our case. The latter VSL sub-profile, provides a concrete textual language for specifying the values of metrics, constraints, properties, and parameters related to performance, in our particular case.

VSL expressions are used in Spark-profiled models with two main goals: (i) to specify the values of the NFP in the model (i.e., to specify the input parameters) and (ii) to specify the metric/s that will be computed for the current model (i.e., to specify the output results). An example VSL expression for a host demand tagged value of type NFP Duration is:

```
(expr=$mapT1, unit=ms, statQ=mean, source=est)
```

This expression specifies that map1 in Figure 3 demands \$mapT1 milliseconds of processing time on average (statQ=mean) and will be obtained from an estimation in the real system (source=est). \$mapT1 is a variable that can be set with concrete values during the analysis of the model.

The initialization of a RDD is described by an initial node in the UML activity diagram. The stereotype SparkWorkloadEvent is used for labelling the initial node. This stereotype captures the essential details of the creation of the RDD. Mainly, the initialization is represented by two parameters. The first one is the sparkPopulation tag. It corresponds to the number of chunks in which the input data is divided when generating the RDD structure. Transformations (SparkMap) and actions (SparkReduce) have independent stereotypes because they are conceptually different, but they inherit from SparkOperation stereotype and, indirectly, from MARTE::GQAM::GaStep stereotype since they are computational steps. In fact, they share the parallelism, or number of concurrent tasks per operation, which is specified by the tag numTasks of the SparkOperation stereotype. Each task has an associated execution time denoted by the tag hostDemand, an attribute inherited from GaStep. The tag OpType specifies the type of Spark operation (i.e., an enumerable SparkOperation=ftransformation,actiong) in case of using the SparkOperation stereotype when modeling UML diagrams. The concept of scheduling in Spark is captured by the stereotype SparkScenario. For simplicity in this first version, we only support a Spark cluster deployed with a static assignation of the resources to Spark jobs (e.g., YARN or standalone modes); and an internal fifo policy (task scheduler). Therefore, the number of CPU cores and memory are statically assigned to the application on launching time. This configuration is reflected in the tags nAssignedCores, nAssignedMemory and sparkDefaultParallelism. They represent respectively the amount of computational cores and memory resources assigned by the scheduler to the current application; and the default parallelism of the cluster configuration. The attribute sparkDefaultParallelism specifies the default number of partitions in a RDD when SparkWorkloadEvent.sparkPopulation is not defined. It also determines the number of partitions returned in a RDD by transformations and actions like count, join or reduceByKey when the value of numTasks is not explicitly set by the user.

The stereotype SparkScenario inherits from MARTE::GQAM::GaScenario. It gathers the rest of the contextual information of the application; for instance, the response time or throughput that will be computed by the simulation. The SparkScenario stereotype is applied to the activity diagram.

Each UML partition is mapped to a computational resource in the UML deployment diagram following the scheduling policy defined for the topology. Figure 4 shows the deployment diagram, which complements the previous activity diagram. Each computational resource is stereotyped as SparkNode (equivalent to GaExecHost) and defines its resource multiplicity, number of cores. This stereotype is inherited from MARTE GQAM.

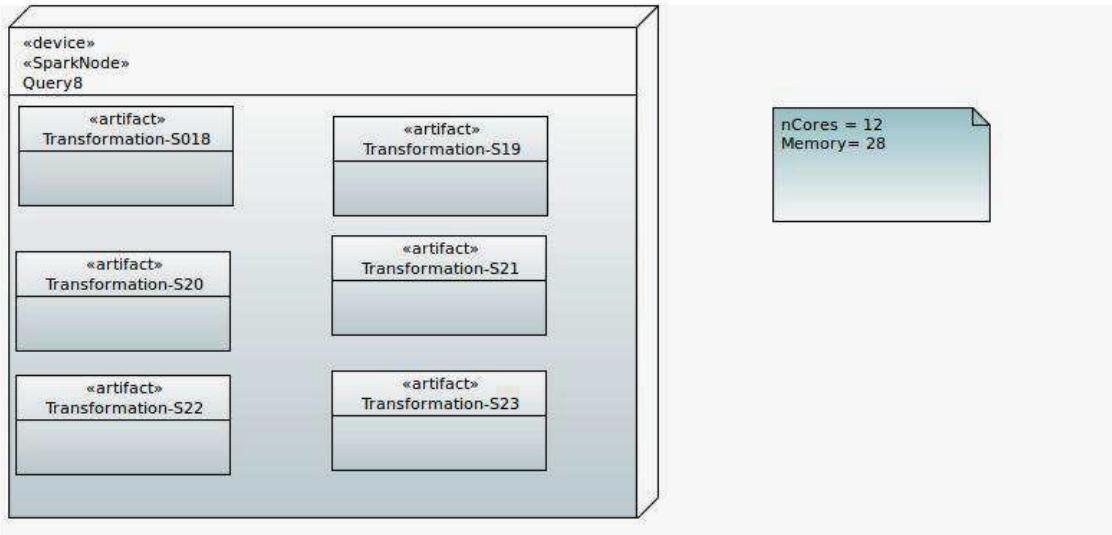


Figure 4. Query8 Deployment diagram

Finally, the SparkNode stereotype is applied over the computational devices in the UML deployment diagram. It represents a resource in the Spark cluster where the tasks are run. The main attributes are the nCores and Memory. The first tag corresponds to the number of available CPUs in the device. The second tag is a boolean that indicates if the size of the partitions in the RDD fit in the memory of the server or they must be stored in a temporal file. The SparkNode stereotype inherits from DICE::DTSM::Core::CoreComputationNode and it is equivalent to the GaExecHost stereotype from MARIE. The Spark profile has been implemented for the Papyrus Modelling environment for Eclipse.

Experimental Results

As an illustrative example, here we report the cost impact analysis of the privacy mechanism for queries 5 and 6 at 1.5 millions dataset with 85s initial deadline. The deadline was iteratively decreased by 20s in order to consider 10 optimization instances. The results are reported in Figure 5. From the experimental results one can see that above 45s and for 20s no extra costs are incurred, while for 25 and 15s deadline the cost overhead due to masking technique is in between 50 and 66%. Deadlines lower than 15s resulted to be too strict and D-SPACE4Cloud did not find any feasible solution. Overall, these results provide a strong point in favour of the optimization procedure implemented in D-SPACE4Cloud, as they prove that making the right choice for deployment can lead to substantial savings throughout the application lifecycle and moreover software architect can take more informed decisions and evaluate a priori the impact of different QoS levels on cloud operation costs.

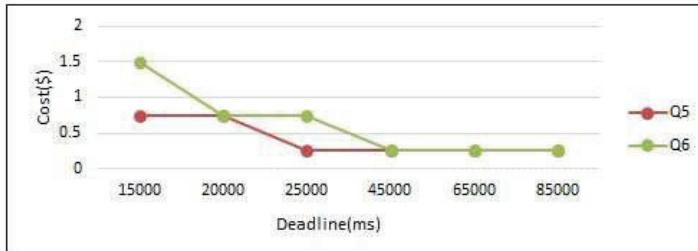


Figure 5. Cost evaluation of Query 5 to 6 by varying deadlines for 1.5 million dataset

From the experimental results one can see that above 45s and for 20s no extra costs are incurred, while for 25 and 15s deadline the cost overhead due to masking technique is in between 50 and 66%. Deadlines lower than 15s resulted to be too strict and D-SPACE4Cloud did not find any feasible solution.

Figure 6 reports the results of for Query 1 and 3, when 10 millions entries data set is considered for performance profiling. The initial deadline is set to 3,500s that is maximum of the execution time of both queries registered on the real cluster and which is iteratively reduced by 500s .

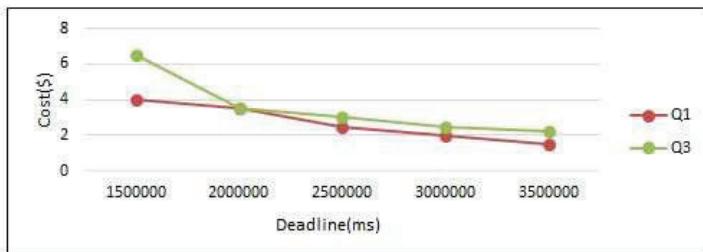


Figure 6. Cost ratio evaluation of Query 1 to 3 by varying deadlines for 10 million dataset

The results show that cost overhead due to masking technique is between 33 and 40%, and no extra costs are incurred for deadlines larger than 2500s. Deadlines lower than 1500s were too strict and no feasible solutions were found.

Further experiments were targeted for encryption. For encryption, we selected data set 10 million and we evaluated Query 7, AES 256 bit encrypted and unencrypted for which we registered the largest performance degradation that is 5%. In this way the results we achieve will be conservative. 80s was set as initial deadline, which was then iteratively reduced by 5s

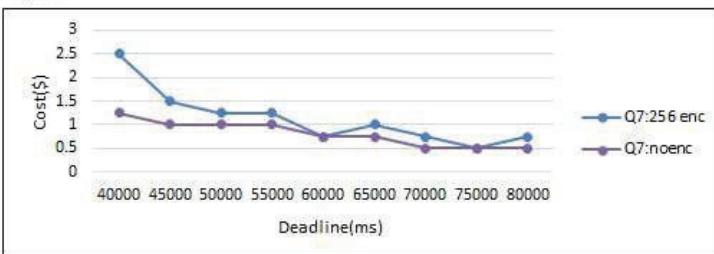


Figure 7. Cost evaluation of Query 7, 256 bit encrypted to unencrypted by varying deadlines for data 10 million.

Results are reported in Figure 7 which shows that 50% cost overhead is achieved at 40s, which is also the minimum deadline that can be supported by the system (otherwise no feasible solution can be found).

Finally, we report the results we achieved by considering the largest dataset, i.e., 30 million, for Query 5. For performance profiling we considered the configuration at 13 nodes which registered the largest performance degradation. 80 was set as initial deadline, which then was iteratively reduced by 20s. Figure 8 reports the cost for AES 256 bit encryption.

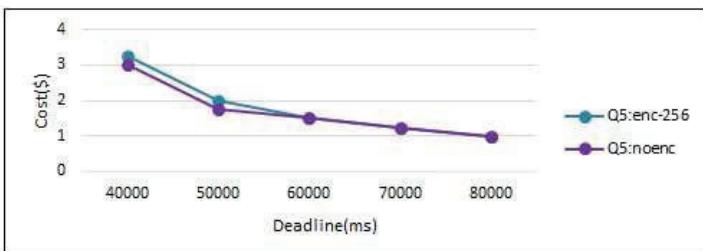


Figure 8. Cost ratio evaluation of Query 5, 256 bit encrypted to unencrypted by varying deadlines for data 30 million

The experiment shows that cost ratio due to encryption is only 13% at maximum. While below 40 seconds D-SPACE4Cloud could not find any feasible solution. Figure 9 reports the results for AES 128 bit encryption for Query 5, the runs were taken using 13 nodes. Here we considered 88,000 milliseconds as initial deadline since it is the largest value measured in real system during experiments concerning to performance degradation.

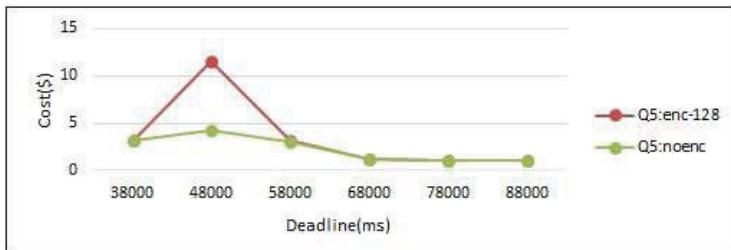


Figure 9. Cost ratio evaluation of Query 5, 128 bit encrypted to unencrypted by varying deadlines for data 30 million

On average encryption cause only 8% overhead on cost and a larger overhead is obtained when the deadline is set to 48s. This is due to an anomalous behaviour of the D-SPACE4Cloud tool, which identifies a local optimal solution and stops its hill climbing algorithm.

Although we performed many experiments, but here we only listed the decisive one. From above experiments results are concluded as masking causes more overhead than encryption, while there is no significant difference between 128 and 256 bit encryption. Overhead on cost due to anonymization was found around 66%, while because of encryption was found only 50% in the very worst case.

Conclusion

Three privacy mechanisms have been considered for NETF case study. In general for NETF case study anonymization causes more overhead than Encryption, two different encryption techniques have negligible difference in terms of effect on performance and cost of system. NETF benchmark has around 66% impact on cost due to masking, while due to encryption the cost increases by around 50%.

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Quality_Optimisation&oldid=3367137'

This page was last edited on 26 January 2018, at 09:47.

Text is available under the [Creative Commons Attribution-ShareAlike License](#). additional terms may apply By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Delivery

Contents

Introduction

Motivation

Existing solutions

How the tool works

TOSCA technology library used in blueprints

Deployment service concepts

Open Challenges

Application domain: known uses

Spark applications

Deployments with Continuous Integration

Conclusion

References

Introduction

In this book so far, we have been focusing on design work, which most of the time happens in our IDE. All the analysis took place in the development space and without running a single line of our code. But the purpose of the design and development is to create an application, which can run in some data centre or on an infrastructure, so that we can use the application's functionality. In this chapter, we will therefore look at the methods and tools that will install, start and test our applications.

In a typical model-driven fashion, a Deployment Model provides an exact representation of our application. Using the [Deployment-Specific Modelling approach](#) is a good start in this direction. A Deployment Model contains all the pieces that make up an application, so we need the tools that can read this model and take steps to set up the application. The expected outcome of these tools is an instantiation of the resources, described in the model, occupied by the services specified to be associated with these resources.

Before getting to the tool capable of deploying application, there is usually one more step: transforming the model into a format that is understandable to the target tool. Instead of the UML representation, tools usually prefer a more compact and targeted language, commonly named as Domain Specific Language (DSL)^[1], which is used to prepare blueprints of DIAs. We will use OASIS TOSCA^[2], a recently emerged industry standard for describing portable and operationally manageable cloud applications. One of its benefits is that it enables flexible use and ability to extend the available nomenclature to fit a specific domain such as Big Data.

The tool we will use for deploying our DIA needs to offer a simple access to managing deployments, and needs to provide repeatable and reliable DIA deployment operations.

Motivation

A DIA is composed of a number of support services and technologies, on top of which we normally run some custom user code. Complexity of the DIA's setup is a function of the needs of the application itself. But regardless of whether we have a simple application, which only stores the data in a NoSQL storage, or a multi-tiered stack of building blocks, someone or something needs to install and start all of the components. In the DevOps setting, we use a cloud orchestration tool^[3] to automate this process.

Individual open source services and tools come with extensive instructions on how to set up and configure the services. Doing this manually requires time and skill, which is often missing in the teams who are starting up in the world of Data-Intensive Application development. The need to learn to deploy services runs mostly orthogonally to the need to learn to make the most of the service itself. Therefore any help to speed up or entirely replace the deployment steps can be a major time and cost saving factor.

The need to automate the deployment process in DevOps has brought a concept of Infrastructure as Code^[4], which enables to store blueprints for whole platforms in a version-oriented repository such as GIT. A great benefit of this is that we have a single source of truth for what the DIA should be like when deployed.

Blueprints written in TOSCA YAML can contain any number of details of the individual components that comprise the DIA. However, like in a traditional application source code, the blueprints too should not repeat concepts that can be represented at a lower level of abstraction. These can be packaged into a TOSCA library, which can be imported into each blueprint as a plug-in.

Existing solutions

Automation of configuration and deployment is not new, because various forms of batch scripts have existed for decades. This is traditionally an imperative approach to configuration automation, where a script describes a series of instructions that need to run in order. A more recent, but an already widespread approach is to use a declarative approach, where we focus on describing the desired state of the system.

The best approach to deploying complex systems is to break the process down such that different tools focus on different levels of the system. Starting from the level of the infrastructure, i.e., the computation, network and storage resources, we can pick from a number of configuration management^[5] tools: Chef^[6], Puppet^[7], Ansible^[8], and others. These tools use definitions with names such as cookbooks, playbooks and similar as the source code for installation and configuration processes. A user or an orchestrator would then execute units of these definitions (e.g., recipes of Chef's cookbooks) to transition between states of the system.

For managing the Cloud applications such as Big Data applications, we need a higher level approach, which takes a form of a cloud orchestrator^[3]. Some of the configuration managers such as Ansible or Chef already have some orchestration capabilities. This ensures that interdependencies are set up in a proper order and configured to properly discover each other. For instance, a web application needs a web server to be configured first, and a database set up and running on another node. This is done by the orchestrator tools, representatives of which include Ubuntu Juju^[9], Apache Brooklyn^[10] and Cloudify^[11].

The tools are only one part of the delivery automation. The other crucial part are the recipes, playbooks, plug-ins and blueprint type definitions. Each tool has a community of vendor's own or community's contributions to the [uge](#) repository or marketplace of the code for configuration and orchestration. The quality of the material available in this way varies, and the compatibility of different recipes is not always guaranteed. But this is a valuable source for initial experiments with technologies and prototyping, as long as we have the time to study each cookbook or playbook.

How the tool works

TOSCA technology library used in blueprints

The DICE delivery process aims to make the interaction with the orchestration as simple as possible. We start off with an application blueprint in a TOSCA YAML format. We can write this document ourselves, but it is even better to follow the DICE methodology and transform a Deployment-Specific Model using DICER into an equivalent XML blueprint. In each case, the blueprint will look similar to this example:

```
tosca_definitions_version : cloudify_dsl_1_3
imports :
  - https://github.com/dice-project/DICE-Deployment-Cloudify/releases/download/0.7.2/full.yaml
node_templates :
  mongo_fw:
    type: dice.firewall_rules.mongo.Common
  standalone_vm:
    type: dice.hosts.ubuntu.Medium
    relationships :
      - type: dice.relationships.ProtectedBy
        target: mongo_fw
  standalone_mongo:
    type: dice.components.mongo.Server
    properties :
      monitoring :
        enabled: true
    relationships :
      - type: dice.relationships.ContainedIn
        target: standalone_vm
  accounts_db:
    type: dice.components.mongo.DB
    properties :
      name: accounts
    relationships :
      - type: dice.relationships.ContainedIn
        target: standalone_mongo
outputs:
  mongo_access:
    description: Mongo client connection details
    value:
      concat:
```

```

- "mongo --host "
- get_attribute : [ standalone_mongo , fqdn ]
- :27017

```

This blueprint is composed of node templates, the types of which are based on the definitions in the DICE Technology Library for example:

- dice.hosts.ubuntu.Medium represents a compute host of a medium size.
- dice.firewall_rules.mongo.Common a node type for defining a networking security group or firewall, such that only the ports needed for MongoDB to communicate are accessible, and this includes peer engine services and any clients.
- dice.components.mongo.Server a component containing all the MongoDB-related modules that comprise a stand-alone instance of the MongoDB engine.
- dice.components.mongo.DB represents a database in a MongoDB engine.
- dice.components.mongo.User a user in a MongoDB cluster

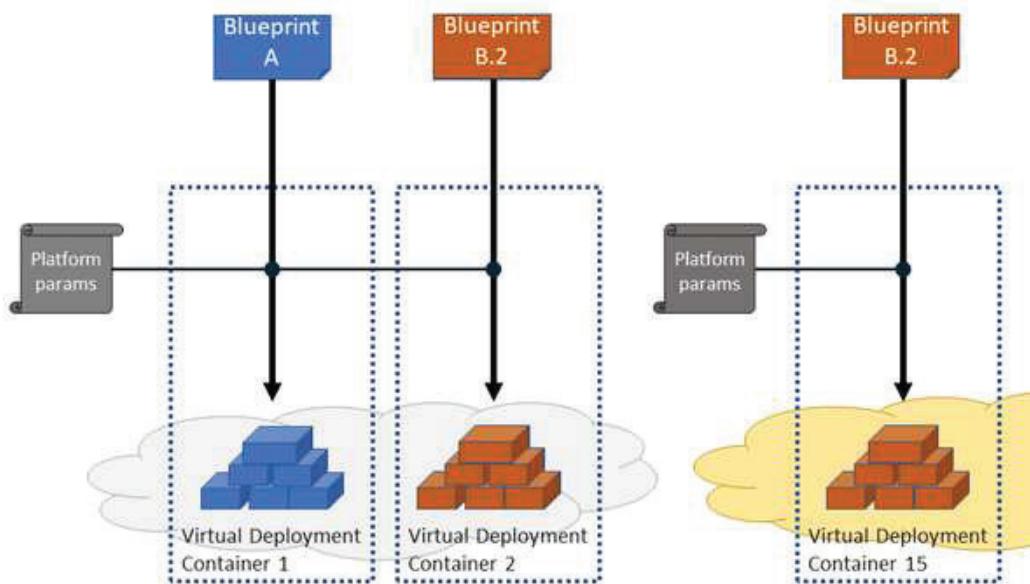
Many of the node templates need to be in a relationship with another node template. **Do this using the following relationship types:**

- dice.relationships.ProtectedBy the source of this relationship is a compute host, and the target is a dice.firewall_rules node template defining the secure group or a firewall.
- dice.relationships.ContainerIn may connect a service-related node template to a compute host, or a database to a database engine such as MongoDB.
- dice.relationships.mongo.HasRightsToUse enables permission of the source user node template with the target database node template.

[More blueprint examples](#) are available.

Deployment service concepts

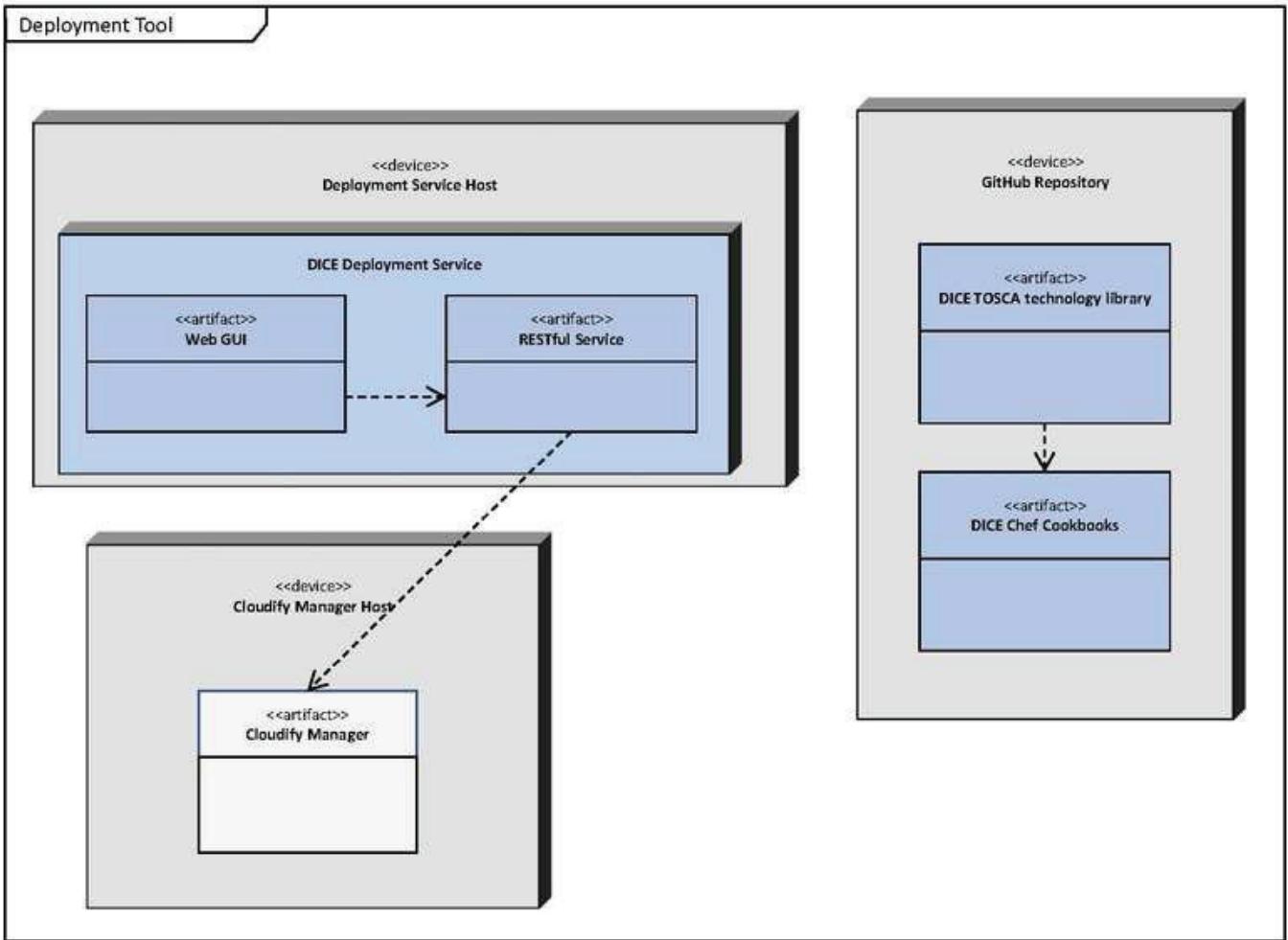
A deployment is an instantiation of a blueprint in the target infrastructure. The cloud orchestrator can generate one or more concurrent or serial (i.e., the previous one is destroyed because a new one is created) deployments. To make the management of deployments simpler, the [DICE Deployment Service](#) operates with the concept of virtual deployment containers. A blueprint submitted to a particular virtual deployment container will result in a deployment associated with that virtual deployment container. A new blueprint submitted to the same virtual deployment container will result in a new deployment that will replace the previous one.



Virtual Deployment Containers concepts for the DICE Deployment Service's

This is illustrated in the figure above, where Blueprint B.1 has previously been deployed in Virtual Deployment Container 2. But then the users improved the application, resulting in the Blueprint B.2. After submitting this blueprint to Virtual Deployment Container 2, the previous deployment has been removed and a new one gets installed. This change leaves the deployment in Virtual Deployment Container 1 intact. Users can create as many containers as needed, for instance to use for personal experimentation of a new technology, specific branches in Continuous Integration, or for manual acceptance testing of new releases.

Another feature of the [DICE Deployment Service](#) and the DICE TOSCA technology library is that it enables deployment to any of the supported cloud infrastructures. In the blueprint, the users can specify the type of the platform to deploy into (e.g., OpenStack or Amazon EC2), and this can be different for different components in the blueprint. For the nodes that do not explicitly specify the target platform, the DICE Deployment Service uses its default target platform. In this case, the same blueprint can be submitted to different cloud providers without any change in the blueprint. On the figure above, we submitted the Blueprint B2 to two different cloud providers. As shown, any platform-specific input parameters are supplied by the DICE Deployment Service. It is up to the administrator of the data centre to set these to the proper values. The designers and developers, on the other hand, do not need to handle such specifics.



Deployment diagram depicting components of the DICE Deployment Service

Behind the scenes, the actual blueprint deployment gets submitted to a RESTful service, which augments the blueprint with any platform-specific details. The actual blueprint orchestration is then performed by Cloudify^[11], which takes care of pulling the relevant elements of the technology library and Chef cookbooks before executing them in the deployment workflow. The figure above illustrates the architecture of the service that enables this workflow.

Open Challenges

The combination of the presented deployment service and the TOSCA technology library highly speeds up preparation and setup of the applications, which are composed of the supported technologies. However, many of the data intensive applications will combine other first party and third party components than the one supported in the library.

The TOSCA technology library provides a short-cut for certain cases of custom elements by providing a node type for custom (bash or Python) scripts. But these are available only for really simple customizations. Because of their over-simplicity, using them does not provide the full functionality of the TOSCA blueprints. The result is that a larger part of such blueprints become imperative than it would otherwise be necessary. A cleaner way to address such issues is to apply extensions and changes at the library level, but this requires a higher degree of skill and knowledge on the user part.

The main strength of the DICE delivery toolset is that it provides deployment of open source solutions as a collection of easy-to-use building blocks. But at the same time this represents a weakness, because the definitions in the library are only as up-to-date as their providers keep them. Since this is not the service's original vendor, the adopters face a risk of lagging behind the development of the mainstream Big Data services.

Application domain: known uses

The DICE Deployment Service in combination with the DICE TOSCA Technology Library may be used in a wide variety of workflows. It also readily caters for a wide variety of Big Data applications. Here are a few of the suggested uses.

Spark applications

The DICE TOSCA technology library contains building blocks, which include support for Apache Spark^[12], Apache Storm^[13], Hadoop^[14] File System, Hadoop Yarn, Kafka^[15], Apache Zookeeper^[16], Apache Cassandra^[17] and MongoDB^[18]. This means that any application, which requires a subset of these technologies, can be expressed in a TOSCA blueprint and deployed automatically

As an illustrative example, a [Storm application blueprint](#) shows how all the necessary ingredients can be represented together:

- The goal is to deploy and run a .jar containing a compiled Java code of a [WordCount topology](#). This is encapsulated in the wordcount node template.
- The platform to execute this program consists of the Storm nodes: a Storm Nimbus node (represented by the nimbus node template) and the Storm Workers (the storm node templates).
- For the Storm to work, we also need Zookeeper in the cluster. The blueprint defines it as the zookeeper node template.
- All of these services need dedicated compute resources to run. The blueprint defines the ports to be open for the services to work: zookeeper_security_group, nimbus_security_group, storm_security_group. For convenience, we also publish Zookeeper and Storm Nimbus on a public network address: zookeeper_floating_ip and nimbus_floating_ip.

Deployments with Continuous Integration

The DICE Deployment Service is a web service, but it also comes with a [command line tool](#). This enables a simple use from Continuous Integration engines. Here we show an example of a Jenkins^[19] pipeline^[20], which builds and deploys a Storm application.

```
pipeline {  
    agent any  
  
    stages {  
  
        stage('build-test') {  
            steps {  
                sh 'mvn clean assembly:assembly'  
            }  
        }  
  
        stage('deploy') {  
            steps {  
                sh '''  
                    dice-deploy-cli deploy --config $DDS_CONFIG \  
                    $STORM_APP_CONTAINER \  
                    blueprint.tar.gz  
  
                    dice-deploy-cli wait-deploy --config $DDS_CONFIG \  
                    $STORM_APP_CONTAINER  
                '''  
            }  
        }  
    }  
}
```

Conclusion

The main promise of automated deployment is in making life and jobs of the operators easier. The power of the approach is such that it enables that most of the tasks can be done by itself - without much intervention of the system administrators. This has a tremendous positive impact on the speed of delivery of applications. We could almost talk about the NoOps approach, where the system operator only sets up the services and configures the necessary quotas in the resource pool of the infrastructure. From this point on, the developers or testers can use the delivery tools in a self-service manner.

Moreover, the ability to consistently and repeatably deploy the same application is an important factor in being able to integrate (re)deployment of application into a larger workflow. In particular, it enables Continuous Integration, where an application gets redeployed to the testbed whenever the developers push a new update to the repository branch. The application can also be periodically tested, whereby installation and clean-up are both parts of the workflow. This opens up many possibilities for performing quality testing (as presented later in the book) and integration testing.

References

1. Domain Specific Language(https://en.wikipedia.org/wiki/Domain-specific_language)
2. OASIS TOSCA (https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca)

3. [Orchestration \(computing\)](https://en.wikipedia.org/wiki/Orchestration_(computing))
 4. [Infrastructure as a service manifest](http://www.infrastructures.org/)
 5. [Configuration Management](https://en.wikipedia.org/wiki/Configuration_management)
 6. [Chef](https://www.chef.io/chef/)
 7. [Puppet](https://puppet.com/)
 8. [Ansible](https://www.ansible.com/)
 9. [Ubuntu Juju](http://www.ubuntu.com/cloud/tools/juju)
 10. [Apache Brooklyn](https://brooklyn.incubator.apache.org/)
 11. [Cloudify](http://getcloudify.org/)
 12. [Apache Spark](https://spark.apache.org/)
 13. [Apache Storm](https://storm.apache.org/)
 14. [Hadoop](https://hadoop.apache.org/)
 15. [Kafka](https://kafka.apache.org/)
 16. [Apache Zookeeper](https://zookeeper.apache.org/)
 17. [Apache Cassandra](https://cassandra.apache.org/)
 18. [MongoDB](https://www.mongodb.com/)
 19. [Jenkins](https://jenkins.io)
 20. [Jenkins Pipeline](https://jenkins.io/doc/book/pipeline/)
-

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Delivery&oldid=3367973'

This page was last edited on 28 January 2018, at 15:42.

Text is available under the [Creative Commons Attribution-ShareAlike License](#). additional terms may apply By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#)

Practical DevOps for Big Data/Configuration Optimisation

Contents

Introduction

Motivation

Existing Solutions

How the tool works

CO methodological steps

Open challenges

Application domains

Storm configuration

Cassandra configuration

Conclusion

References

Introduction

Finding optimal configurations for data-intensive applications is a challenging problem due to the large number of parameters that can influence their performance and the lack of analytical models to anticipate the effect of a change. To tackle this issue, DevOps can incorporate tuning methods where an experimenter is given a limited budget of experiments and needs to carefully allocate this budget to find optimal configurations. In this section we discuss a methodology that could be used to implement configuration optimization in DevOps.

Motivation

Once a data-intensive system approaches its final stages of deployment for shipping to the end users it becomes increasingly important to tune its performance and reliability. Tuning can be also important in initial prototypes, to determine bounds on achievable performance. In either case, this is a time-consuming operation since the number of candidate configurations can grow very large. For example, the optimization of 10 configuration options taking ON/OFF values would require in the worst case 1024 experiments to determine the globally optimum configuration. As data-intensive applications may feature multiple technologies simultaneously, for example a data analysis pipeline consisting of Apache Spark and Kafka, it becomes increasingly difficult to optimally configure such systems. Sub-optimal configurations imply that a technology is more expensive to operate and may not deliver the intended performance, forcing the service provider to supply additional hardware resources. Configuration optimization tools are therefore sought in DevOps to guide this configuration phase in order to *quickly* find a configuration that is reasonably close to optimal.

Existing Solutions

There are currently a number of mathematical methods and approaches that can be used in the search for optimal configuration.

A classic family of mathematical methods consists of design of experiments (DoE) methods, which fit a nonlinear polynomial to response data to predict system response in unexplored configurations. Such methods provide a way to reduce the number of experiments needed to fit the response model in particular cases, such as those where configuration options are binary valued (e.g., ON/OFF). It is however difficult to consider tens of parameters with these methods, which are often limited to much smaller dimensionality (8-10).

Related methods are those developed in statistics and machine learning to address the multi-armed bandit problem. This is a general problem that abstracts allocation of a finite set of resources with incomplete information on the rewards that can be obtained from the decisions. Such situations arise commonly in configuration optimization, where it is not clear in advance how changing the level of a configuration option will impact the system. The crucial trade-off is here to decide what amount of time to spend for exploitation vs exploration, while the former refers to focusing on the optimization option with the largest expected payoff, while the latter refers to improving the knowledge of the other optimization options to refine the estimates of the expected payoffs. The method proposed in this chapter, called Bayesian optimization, belongs to this class of solution techniques and offers flexibility in choosing such trade-off.

Several commercial services for performance optimization exist in the market, e.g.:

- PragmaticWorks^[1]
- Oracle^[2]
- HP^[3]
- Centaurea^[4]

These services in part depend on the expertise and skills of consultants. The solution that is pursued here is algorithmic and very few comparable methods exist for Big Data frameworks. For example, <http://unraveldata.com> offers a Big Data management solution involving auto-tuning for Big Data applications. No public information exists on the underpinning auto-tuning methods, and the website is rather vague, suggesting that this is not the main feature of the product. Conversely, the solution discussed here is entirely tailored to configuration and features innovative methods based on Gaussian processes that have already been proved experimentally more effective than the scientific state-of-the-art.

How the tool works

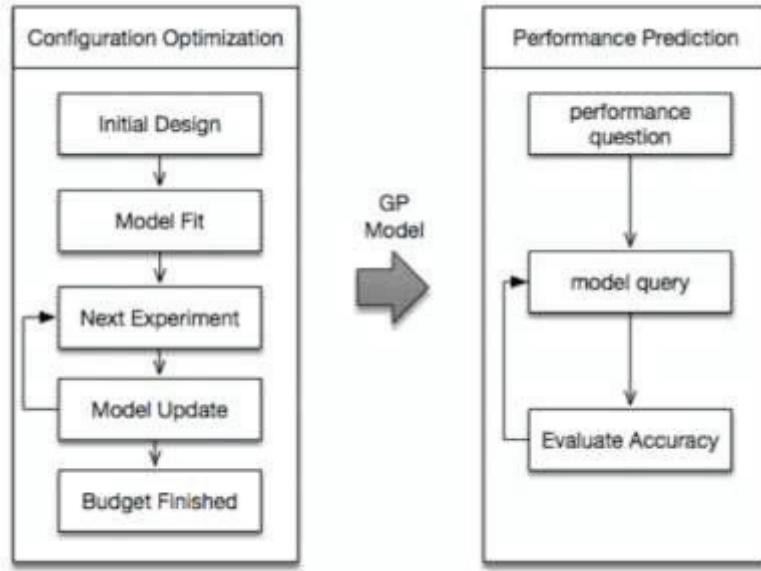
Configuration optimization (CO) uses a DevOps toolchain to iteratively run tests on the Big Data application. At each iteration, a new configuration is tested using a load generator and performance data is collected. This data is used to train machine learning models to decide the configuration to evaluate the system at the next iteration. Using this decision, the testing cycle is iterated, until a configuration that achieves performance objectives is found.

The challenge for a testing system is to define an efficient search algorithm, which can automatically decide what experiments to run in order to quickly tune the system. The fewer the experiments are needed to tune the system, the more cost-effective the process will be. The main issue is that the system behavior is hard to predict, thus the influence of a configuration parameter on performance may be not be known before load tests are carried out on the application after that parameter is changed.

BO4CO is an implementation of CO produced in DICE that focuses on optimizing configurations based on a technique known as Bayesian Optimization. Bayesian Optimization is a machine learning methodology for black-box global optimization. In this methodology, the unknown response of a system to a new configuration is modeled using a Gaussian process (GP), an important class of machine learning models. GPs are used to predict the response of the platform to changes in configuration. GPs can take into account mean and confidence intervals associated with measurements, predict system behavior in unexplored configurations, and can be re-trained quickly to accommodate for new data. More importantly, Bayesian optimization method can be much faster on real systems than any existing auto-tuning technique.

CO methodological steps

The logic of the CO tool is iterative. The configuration optimizer automatically selects a configuration at each iteration employing the BO4CO algorithm that determines the best configuration to test next during the procedure. BO4CO estimates the response surface of the system using observed performance data. It selects the next configuration to test, using the estimated data, searching for points that have a good chance of being the optimum configuration. This is illustrated on the left-hand side of the figure below, which shows these exact steps as defining the configuration optimization process.



The right-hand side of the above figure recalls the important fact that the response models based on GPs provide the ability to query the performance of the application in untested configurations, thus providing a mechanism to guide the CO search. In particular, after making a new test it is possible to assess the accuracy of the model and supply a new observation to it, to refine its accuracy for future use. In the DICE implementation of the methodology, there is tool support to automatically perform training and prediction with GPs, in addition to a GUI-based specification of the boundaries of the configuration options.

Open challenges

A methodology such as CO could be applied in principle also to migration problems, but at present it has only been used in the context of optimizing a new data-intensive application. Many companies and public sector organizations are progressively migrating their applications to emerging Big Data frameworks such as Spark, Cassandra, Storm, and Hadoop. Three research challenges may be envisioned:

1. Auto-tuning methods based on machine learning are not customized for individual Big Data technologies, leading to sub-optimal experimental time and cost, which are worsened especially when multiple technologies hosted are used together.
2. Performance auto-tuning lacks validation in industrial Big Data context, current research focuses on academic and lab testbeds.
3. Big Data require continuous auto-tuning, since workload intensities & data volumes grow continuously

Application domains

Storm configuration

The CO methodology has been tested in [5] against different Apache Storm topologies. The improvements to the Storm topology configuration and to the end latency (the difference between the timestamp once the job arrives at the topology and the time it has been processed and leaves the topology) when CO finds optimal configuration comparing with the default values are up to 3 orders of magnitudes compared with the default values. The study shows that the tool finds the optimum configuration only within first 100 iterations. The full factorial combination of the configuration parameters in this experiment is 3840, and 100 experiments is equal to

2% of the total experiments. Note that in order to measure the difference between the optimum configuration found by the configuration optimization tool we performed the full factorial experiments (i.e., 3840 experiments each for 8 minutes over $3480*8/60/24=21$ days). Further application to a real-world social media platform can be found [6].

Cassandra configuration

A validation study has also been performed in [6] on optimizing Cassandra read latency. A variant of the BO4CO algorithm called TL4CO has been also considered. TL4CO integrates into the Bayesian optimization method a technique called transfer learning, which allows re-using historical data from past configuration optimization cycle to accelerate new configuration cycles. More details about TL4CO can be found in [6]. The experiments collected measurements of latency versus throughput for both read and write operations in a 20-parameter configuration space. The configurations that are found by the CO tool (with TL4CO and BO4CO algorithms), the default settings and the one prescribed by experts are annotated. The results have shown that the configuration that the CO tool with TL4CO initialization finds only after 20 iterations results in a slightly lower latency but much higher throughput compared to the one suggested by the experts.

Conclusion

This chapter has discussed the problem of automatically optimizing the configuration of data-intensive applications. Technologies such as Storm exemplify the challenge, as they require to jointly optimize several tens of configuration parameters without a good understanding on how each parameter level interacts with the other parameters. The CO DevOps tool illustrates a possible solution. Iterative experiments are conducted until finding an optimal configuration. Bayesian optimization, a black-box optimization technique, can considerably accelerate the configuration task compared to standard experimental methods from statistics, such as response surfaces and experimental designs.

References

1. PragmaticWorks (<http://pragmaticworks.com/Client-Solutions/Architecture-and-Configuration>)
 2. Oracle (<http://www.oracle.com/us/products/consulting/resource-library/bigdata-sql-info-int-rapidstart-2332772.pdf>)
 3. HP (<https://www.hpe.com/us/en/services/consulting/big-data.html>)
 4. Centaurea (<http://centaurea.io/services/it-consulting>)
 5. Jamshidi, Pooyan; Casale, Giuliano (2016)."An Uncertainty-Aware Approach to Optimal Configuration of Stream Processing Systems". IEEE MASCOTS, IEEE Press, <http://ieeexplore.ieee.org/document/7774564/>
 6. "Deliverable 5.2. DICE delivery tools – Intermediate version"2017. http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2017/02/D5.2_DICE-delivery-tools-%E2%80%93-Intermediate-version.pdf
-

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Configuration_Optimisation&oldid=3367974

This page was last edited on 28 January 2018, at 15:42.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Quality Testing

Contents

Introduction

Motivation

Existing Solutions

How the tool works

QT-LIB

QT-GEN

QT Methodology

Open Challenges

Application domain: known uses

Generating an artificial Twitter trace for load testing

Load testing a Kafka pipeline

Conclusion

References

Introduction

Quality testing (QT) of data-intensive application aims at verifying that prototypes of a DIA deliver the required level of scalability, efficiency, and robustness expected by the end-user. Common Big data technologies such as Apache Storm, Spark, Hadoop, Cassandra, and MongoDB are fairly different from each other, thus it is important to realize that QT requires using multiple tools to test a complex DIA.

For example, Cassandra and MongoDB databases recently became supported by the Apache JMeter – a state-of-the-art load generation tool in the open source domain. MongoDB is supported natively, whereas Cassandra is supported through an external plugin. There is also partial support for Apache Hadoop and HBase, now available in JMeter. This means that the research challenges associated with the load injection for these technologies are limited since a practical solution already exists, being JMeter the leading open source solution for stress testing. Based on the above, we discuss in this chapter mainly QT in the scope of stream-based DIAs, which has received limited attention in the open source community

Motivation

Several challenges arise in designing quality testing tools^[1]. There is no formalized methodology in this respect for DIAs and some of the key challenges in our view may be summarized as follows.

- **Load injection actuator design** In traditional load injection, the workload submission relies on a client-server model where external worker threads submit jobs to the application. This model is however not natural for platforms like Storm, where the message injection is done by dedicated software components, called spouts, that are part of the application itself, and which inject specific units of work packaged as messages that flow through the system. In systems of this kind, the load injection actuators should not be implemented as external elements, but rather as internal elements to the application. Therefore, there is a conceptual difference between developing QT tools for databases and similar tools for streaming platform. For example, being internal to the application itself, the resource usage of the workload generator in a streaming application will cumulate to the resource consumption of the application itself. This may not be acceptable for database systems, but it's necessary for streaming applications, where the injection of messages consumes a separate pool of resources. While also an external implementation may be considered, for example by providing a service equivalent to the ones used by Twitter to push data to

external parties, a considerable advantage of the internal implementation is that the load injection actuator can have direct access to the data structures used within the DIA, including the custom message structure defined by the developer. This makes much simpler for the end user to define the streaming workload sent to the system using the same Java classes developed for the DIA itself and to automate the execution of the experiments. However one important implication of this approach is that load injection needs to be provided as an embedded library that can be linked in the application code, with the developer defining the test workload through the IDE.

- **Scalability testing** A requirement for any load injection tool is to be able to test the ability of a system to scale, i.e., to deliver Quality-of-Service (QoS) in terms of performance and scalability also when the parallelism level of the workload is increased (e.g., more users, greater data velocity/greater data volumes, etc.). It is therefore expected that a good implementation of the QT tools will be able to generate additional load as required by the end user until observing a saturation of one or more resources. This is possible with QT
- **Streaming workload generation** Streaming-based DIAs typically focus on analyzing incoming messages for specific topics and trigger certain actions when pre-defined conditions are met on the payload of the messages. One research challenge is, therefore, to provide adequate tool support to make it easy for the user to generate time-varying workloads and time-varying message contents. Ideally such workloads should resemble as much as possible the real workloads seen by the developer's company in production. For example, in the case of Twitter streaming data, one would like to generate workloads that are similar to ordinary Twitter feeds, but that are controllable in terms of arrival rate of individual kinds of messages (e.g., arrival rate of original tweets vs. arrival rate of retweets, or arrival rates of messages from a specific user) and frequency of appearance of messages that trigger specific actions from the DIA. We have systematically investigated this problem throughout the research activity part of QT, and we have developed a dedicated mathematical theory to address this problem.

Existing Solutions

There is a variety of load testing tools available for distributed systems, both commercial (e.g., IBM Rational, HP LoadRunner, Quali, Parasoft, etc) and open-source (e.g., Grinder, JMeter, Selenium, etc). However, all such tools have been developed mainly for traditional web applications where the incoming load is web traffic, and users interact with the application front end (website). In the case of DIAs, the incoming data (e.g. databases, streams) are the main workload drivers. While database testing is fairly common, there is a shortage of open source solutions to inject data streams into a DIA. Prior to introducing the DICE solution to this problem, below we discuss in more details existing load testing tools.

- **Grinder**. This is a framework for distributed testing of Java applications. In addition to testing HTTP ports, it can test web services (SOAP/REST), and services exposed via protocols such as JMS, JDBC, SMTP/FTP and RMI. The tool executes test scripts written in Java, Jython or Clojure. Tests can be dynamic, in the sense that the script can decide the next action to perform based on the outcome of the previous one. The tool is mature, having been first released in 2003. It is quite popular with weekly download count of around 400 downloads. A limitation of Grinder is the lack of an organised community of volunteers to support this tool.
- **Apache JMeter**. This is a distributed load testing tool that offers similar features to Grinder in terms of load injection capabilities. If combined with Markov4JMeter the tool can also feature probabilistic behaviour. Interestingly, the tool also features native support for MongoDB (a NoSQL database). The tool excels in extensibility, which is achieved through plug-ins. A limitation of JMeter is that the web navigation may not be representative on websites where Javascript/AJAX play a major role for performance, since JMeter does not feature the same range of Javascript capabilities as a complex browser. JMeter has support of the large Apache Foundation community.
- **Selenium**. This is a popular library that automates the control of a web browser such as Internet Explorer or Firefox. It allows executing tests using a native Java API. Furthermore, test scripts can be recorded by adding Selenium plug-in to the browser and manually executing the operations. An extension, formerly called SeleniumGrid and now integrated in the stable branch of the main Selenium tool, allows to perform distributed load testing.
- **Chaos Monkey**. This is an open source service that runs on Amazon Web Services. This tool is used to create failures within Auto Scaling Groups and to help identify failures within cloud applications. The design is flexible and can be expanded to work with additional cloud providers and provide additional functionality. The service has a fully configurable schedule for when actions will be taken and by default runs on non-holiday weekdays between 9am and 3pm. A similar tool has been developed within .NET for Microsoft Azure called AZMonkey which offers similar functionality as Chaos Monkey such as rebooting or reimaging role instances within a given Azure deployment at random.

How the tool works

The DICE QT tool is the combination of two independent modules:

- QT-LIB: a Java library to define load injection experiments in Storm-based and Kafka-based applications.
- QT-GEN: a tool for the generation of workload traces that can be injected into the system using QT-LIB.

QT-LIB is offered as a jar file that can be packaged in any Java-based Storm topology, providing the custom spouts for load injection and workload modulation. This tool can acquire external data to be pushed to the DIA either through MongoDB, for large datasets which can be exported as JSON files, or through external text files. The JSON file uses syntax compatible with MongoDB. Below, we provide more details on the individual modules.

QT-LIB

The QT-LIB module offers a set of custom spouts that automate workload injection in the DIAs. The conceptual working of these spouts is simple: as soon as the DIA is deployed, these spouts automatically generate tuples according to the user-specified workload. A load generation interface is available to the end user to specify an input data source (e.g., CSV, MongoDB); this interface provides the input tuples for the spout to inject load into the system and can be easily adapted to other databases or other textual or binary input sources. Even though QT-LIB becomes part of the topology under test (both as a code and logical architecture), the system under test is a collection of Bolts that process data. Therefore, the Bolts and the relationships among them are embedded in the DIA code.

QT-GEN

We discuss the most advanced feature of QT, which is the ability to automatically generate artificial workloads from a given trace of Storm messages. The idea is to distinguish a set of message types within a given workload trace and produce a novel trace that has similar characteristics to the initial trace, without however being identical to it, and with arbitrary user-specified arrival rates. QT assumes the availability of a JSON file where each entry represents a message to be serialized into tuples.

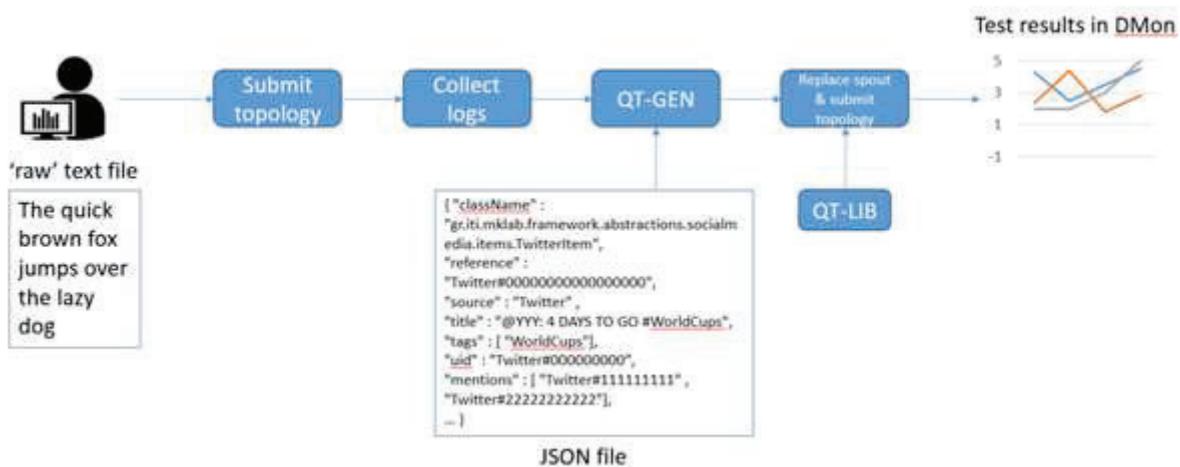
The QT-GEN tool accepts in input a JSON file including an arbitrarily long sequence of such messages, parses it to extract the sequence of issue timestamp and then generates a new sequence of messages where the messages are alternated in such a way to preserve a similar correlation among the user-defined types as and then generates a new sequence of messages where the messages are alternated in such a way to preserve a similar correlation among the user-defined types as seen in the original trace. As a simple example, for a Twitter trace the QT tool can generate a new trace with:

- The same ratio of tweets and retweets as in the original trace, with the latter being identified by a specific field in the Twitter JSON data structure.
- A statistically similar temporal spacing between tweets and retweets, so that volumes of tweets and retweets and the frequency and correlation of their peaks are also reproduced in the artificial workload.

The QT tool can statistically analyze this correlation and reproduce it to the best possible extent in artificial workloads, using a class of traffic models called Marked Markovian Arrival Processes, which are a special class of Hidden Markov Models. Markovian Arrival Processes (MAPs) are a class of stochastic processes used to model the arrivals from a traffic stream. Marked MAPs (MMAPs) are an extension of MAPs that allow modeling arrivals of multiple types of traffic. The QT-GEN library provides a backend to the QT tool, and it is designed for fitting marked traces with MMAPs and successive generation of representative traces. The latter problem is nontrivial due to the typically complex and nonlinear relationships between the statistical descriptors of the marked trace and the parameter of the MMAPs.

QT Methodology

The overall methodology for quality testing is illustrated in the figure below for a reference Storm-based application:



The main methodological steps are as follows:

1. Submission of an application (i.e., a topology) to the cloud testbed
2. Tests with representative workloads and collection of initial logs (e.g., in JSON format)
3. Submission of the logs to the QTGEN workload generator
4. Production of a new workload to be injected into the system using the custom QT-LIB spout
5. Submission of the QT-enabled topology with the QT-LIB spout
6. Automated test carried out
7. Visualization of the results either on the monitoring platform (D-MON) or in the delivery service (Jenkins dashboard)

Open Challenges

The solution outlined in this chapter takes care of load-injection in a stream processing system with the goal of automatically expose performance and reliability anomalies. Although it is possible to model labeled data, current load-testing solutions used in stream-processing systems do not provide the ability to programmatically specify the data content of the message payloads. Clearly, the response of a stream processing system will also depend on the content of the data itself, thus it is foreseeable that more advanced tools could be defined to extend the proposed approach to encompass content, data concepts, and similar

Another aspect that is not explored in this chapter is the scalability of the quality testing toolchain, which is relevant to data-intensive applications designed to run on tens or hundreds of nodes. Such configuration requires the quality testing process itself to be distributed, in order to avoid bottleneck in input stream generation. Commercial solutions for cloud-based load testing are available, but their use in the context of Big data technologies is still in its infancy

Application domain: known uses

Generating an artificial Twitter trace for load testing

Let us consider a stream processing system that obtains data from a Twitter stream. Naturally, in order to test the system, we need some Twitter data. Unfortunately, Twitter and other social media platforms require a payment to obtain and use their datasets. While we could buy some or replay some toy dataset, it would be desirable to have a free tool to generate artificial datasets similar to the real ones. QT offers this capability

The starting point is to consider the JSON template for messages coming from Twitter or a similar platform. For example, this may be similar to this JSON file:

```
{
  "className": "gr.iti.mklab.framework.abstractions.socialmedia.items.TwitterItem",
  "reference": "Twitter#0000000000000000",
  "source": "Twitter",
  "title": "@YYY: 4 DAYS TO GO #WorldCups",
  "tags": ["WorldCups"],
  "uid": "Twitter#0000000000"
}
```

```

"mentions" : [ "Twitter#11111111" , "Twitter#2222222222" ],
"referencedUserId" : "Twitter#138372303",
"pageUrl" : "https://twitter.com/CinemaMag/statuses/123456789123456789" ,
"links" : [ "http://fifa.to/xyxyxxy" ],
"publicationTime" : 1401310121000 ,
"insertionTime" : 0,
"mediaIds" : [ "Twitter#123456789123456789" ],
"language" : "en",
"original" : false,
"likes" : 0,
"shares" : 0,
"comments" : 0,
"_id" : "Twitter#123456789123456789" }

```

Note in particular that the JSON file contains a timestamp field, here "publicationTime", and several fields that may be used to classify the type of tweet, e.g., "original" that distinguishes in two categories: tweets from re-tweets.

In the real Twitter-based DIAs, a message like this is received immediately after publication. Our goal is to produce a new JSON trace, starting from a given Twitter trace, with the following properties:

- The arrival rate of tweets in the Twitter trace and the one in the new trace is identical
- The artificial and original traces have similar probability of generating a burst of tweets

The QT-GEN tool ensures these two properties. Given the trace, the timestamp field (e.g., "publicationTime") and the classification field (e.g., "original"), it outputs a new JSON trace similar to the original Twitter trace, but nonetheless non-identical.

Load testing a Kafka pipeline

We now illustrate the practical use of QT-LIB with Kafka. Below we present a compact [example](#) that is included in the QT-LIB distribution. Initially, as done also with Storm, we construct a QTLoadInjector factory that will assemble the load injector. Moreover, we specify the name of the input trace file, which is assumed to be packaged within the jar file of the data-intensive application.

```

package com.github.diceproject.qt.examples ;

import com.github.diceproject.qt.QTLoadInjector ;
import com.github.diceproject.qt.producer.RateProducer ;

public class KafkaRateProducer {
    public static void main(String[] args) {
        QTLoadInjector QT = new QTLoadInjector () ;
        String input_file = "test.json" ; // this is assumed to be a resource of the project jar file
    }
}

```

We are now ready to instantiate a QT load injector for Kafka, which is called a RateProducer:

```

RateProducer RP;
RP=QT.getRateProducer();
String topic = "dice3"; // topic get created automatically
String bootstrap_server = "localhost:9092";

```

Here we have assumed that our target topic is called *dice3* and it is exposed by a Kafka instance available on the local machine on port 9092. This corresponds to the port of the so-called Kafka bootstrap server, and should not be confused with the port of the Zookeeper instance associated to Kafka. We can now use the run() method to start immediately the load testing experiment.

```

RP.setMessageCount (101);
RP.run(bootstrap_server , topic, input_file);

```

These will read JSON messages similar to the ones created by QT-GEN from the *test.json* file previously declared. This file must be shipped within the application JAR, so that QT-GEN can open it as a local resource. The examples generated 101 message to illustrate the behavior of QT-LIB when the trace runs over since in this case the trace is composed of 100 messages. QT-LIB will cyclically restart the trace from the beginning, resending the first JSON message as the 101st message.

Conclusion

The main achievements of DICE QT are as follows:

- DICE QT allows load injection in Storm-based applications using custom spouts that can reproduce tabulated arrival rates of messages or inject in controlled manner messages stored in a text file or an external database, i.e., MongoDB.
- DICE QT can analyze a workload trace composed of different types of messages and fit the observed inter-arrival times of these messages into a special class of Hidden Markov Models, called MMAPs, from which new statistically similar traces can be generated to assess the system under varying load scenarios.
- DICE QT can also inject load in Kafka topics, therefore supporting a broad variety of targets, including Spark applications

References

1. André B. Bondi (2015). *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability and Practice*. Addison-Wesley.
-

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Quality_Testing&oldid=3367968'

This page was last edited on 28 January 2018, at 15:40.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Fault Injection

Contents

Introduction

Motivation

Existing solutions

How the FIT works

Design

Operation

Graphical User Interface

Installation

Open Challenges

Application domain

Integration

Validation

Conclusion

References

Introduction

The operation of data intensive applications almost always requires dealing with various failures. Therefore during the development of an application, tests have to be made in order to assess the reliability and resilience of the system. These test the ability of a system to cope with faults and to highlight any vulnerable areas. The fault-injection tool (FIT) [1] allows users to generate faults on their Virtual Machines, giving them a means to test the resiliency of their installation. Using this approach the designers can use robust testing, highlighting vulnerable areas to inspect before it reaches a commercial environment. Users or application owners can test and understand their application design or deployment in the event of a cloud failure or outage, thus allowing for the mitigation of risk in advance of a cloud based deployment.

Motivation

Current and projected growth in the big data market provides three distinct targets for the tool. Data Centre owners, cloud service providers and application owners are all potential beneficiaries due to their data intensive requirements. The resilience of the underlying infrastructure is crucial to these areas. Data Centre owners can gauge the stress levels of different parts of their infrastructure and thus offer advice to their customers, address bottlenecks or even adapt the pricing of various levels of assurances. For developers FIT provides the missing and essential service of evaluating the resiliency and dependability of their applications, which can only be demonstrated in the application's runtime by deliberately introducing faults. By designing the FIT to be a lightweight and versatile tool it is trivial to use it during Continuous Integration or within another tool for running complex failure scenarios. Used in conjunction with other tools not within the scope of this report, FIT could monitor and evaluate the effect of various faults on an application and provide feedback to the developers on application design.

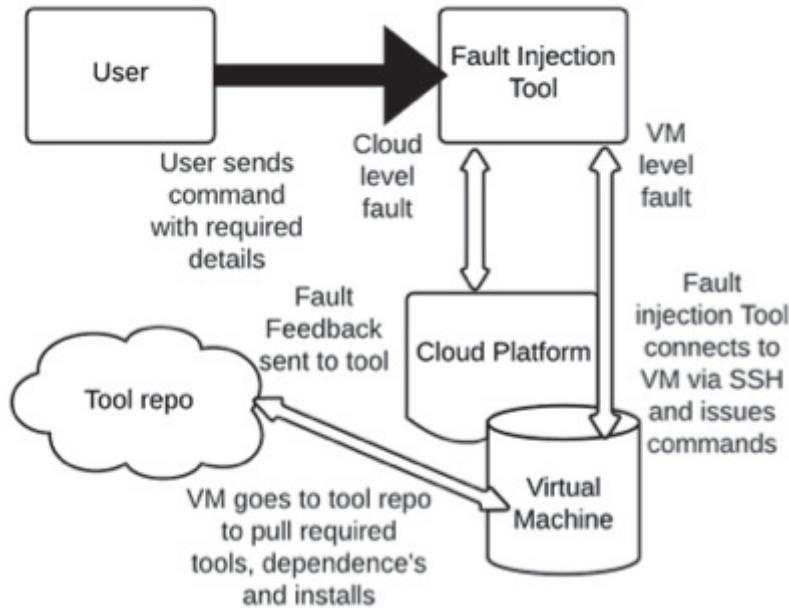
Existing solutions

- DOCTOR (IntegrateD SOftware Fault InjeCTOn EnviRonment)^[2] allows injection of memory and register faults, as well as network communication faults. It uses a combination of time-out, trap and code modification. Time-out triggers inject transient memory faults and traps inject transient emulated hardware failures, such as register corruption. Code modification is used to inject permanent faults.
- Orchestra^[3] is a script driven fault injector which is based around Network Level Fault Injection. Its primary use is the evaluation and validation of the fault-tolerance and timing characteristics of distributed protocols. Orchestra was initially developed for the Mach Operating System and uses certain features of this platform to compensate for latencies introduced by the fault injector. It has also been successfully ported to other operating systems.
- Xception^[4] is designed to take advantage of the advanced debugging features available on many modern processors. It is written to require no modification of system source and no insertion of software traps, since the processor's exception handling capabilities trigger fault injection. These triggers are based around accesses to specific memory locations. Such accesses could be either for data or fetching instructions. It is therefore possible to accurately reproduce test runs because triggers can be tied to specific events, instead of timeouts.
- Grid-FIT (Grid – Fault Injection Technology)^[5] is a dependability assessment method and tool for assessing Grid services by fault injection. Grid-FIT is derived from an earlier fault injector WS-FIT which was targeted towards Java Web Services implemented using Apache Axis transport. Grid-FIT utilises a novel fault injection mechanism that allows network level fault injection to be used to give a level of control similar to Code Insertion fault injection whilst being less invasive.
- LFI (Library-level Fault Injector)^[6] is an automatic testing tool suite, used to simulate in a controlled testing environment, exceptional situations that programs need to handle at runtime but that are not easy to check via input testing alone. LFI automatically identifies the errors exposed by shared libraries, finds potentially buggy error recovery code in program binaries and injects the desired faults at the boundary between shared libraries and applications.

How the FIT works

Design

The FIT has been designed thus far to specifically to comprise a best practice strategy to ensure data intensive applications are reliable, allowing for rigorous testing of applications both during development and after deployment.



FIT architecture

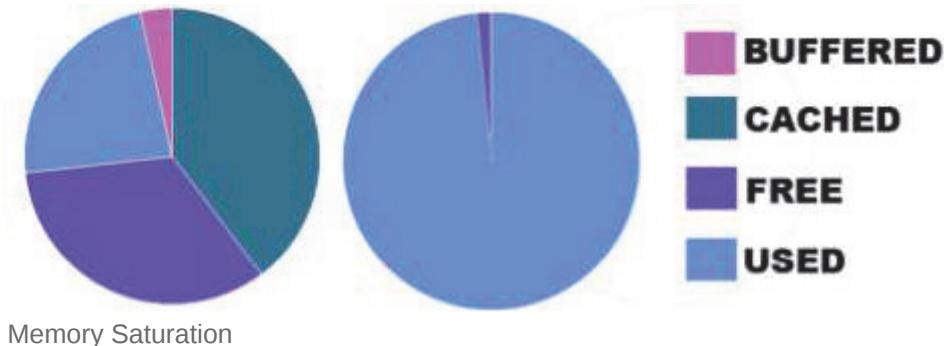
This will allow cloud platform owners/Application VM owners a means to test the resiliency of a cloud installation and applications by generating faults at the cloud level.

The DICE FIT has also been designed and developed in a modular fashion. This allows the replacement of any function that carries out Faults as well as potential to extend the tool as required. The FIT was designed to be as lightweight on the VMs as possible, so for this reason only existing well-tested tools have been implemented for causing faults. The FIT downloads, installs and configures only what is required at the time, because of this no unnecessary tools or dependences are installed. FIT allows VM Admins, application owners and Cloud Admins to generate various faults. It runs independently and externally to any target environment. Figure 1 illustrates the architecture.

Operation

The FIT is designed to work from the command line or through a Graphical User Interface. The user can invoke actions which connect to the target VM and automatically install any required tools and dependences or evoke the required APIs. The command line switches and parameters allow users to select a specific fault and the parameters of the fault such as the amount of RAM to user or which services to stop. An example command line call to connect to a node using SSH and cause memory stress with 2GB is as follows:

```
--stressmem 2 2048m Ubuntu@109.231.126.101 -no home/ubuntu/SSHKEYS/VMKey.key
```



This call was ran on a real cloud system. The tool connected via SSH and determined the OS version by checking the `/etc/*-release`, Ubuntu in this case. It then gathered the memory stress tool suitable for Ubuntu, which is Memtester [7] in this case. Finally the FIT called Memtester to saturate memory on the target node. Figure 2 shows available memory on the target node before (left) and during FIT's invocation (right) as detected by a monitoring tool, where it can be seen that nearly all 2GB of available RAM had been saturated.

To access the VM level and issue commands the DICE FIT uses JSCH to SSH to the Virtual Machines. By using JSCH the tool is able to connect to any VM that has SSH enabled and can then issue commands as a pre-defined user. This allows greater flexibility of commands as well as the installation of tools and dependencies. The DICE FIT is released under the permissive Apache Licence 2.0 [8] and supports the OS configurations Ubuntu (tested with versions 14.04 and 15.10), and Centos with set Repo configured and wget installed (tested on version 7).

Graphical User Interface



Virtual Machine Failures

Deployment Faults	Create faults within a deployment
Block a VM	Block external access to a virtual machine
Stop Random VM	Stop a random VM from a cloud in FCO
Stop a Service	Stop a service running on a virtual machine
Virtual Machine Resource Overload	
CPU Overload	
RAM Overload	
Disk Overload	Overload the CPU, RAM or Disk on a virtual machine

FIT GUI

The GUI provides the same functionality as the command line version of the tool. The GUI provides users with a visual way of interacting with the tool which can make the tool more accessible for a range of users. The user can select from the available actions from a home screen, as seen in Figure 3. Each button leads to a page where a user can enter the relevant inputs and then the fault can be executed. These inputs are the equivalent of the command line parameters.

In the CPU overload page, the user enters the details of the VM and the amount of time to run the overload for. In place of the password the user can also upload an SSH key from a file. Any feedback and output from running the fault is shown at the bottom of the page.

The screenshot shows the DICE Fault Injection Tool GUI. At the top, there's a header with the DICE logo and the text "Fault Injection Tool". Below the header, there's a section titled "Cause high CPU usage". This section contains fields for entering VM details and fault parameters:

- IP Address *: 109.231.122.245
- Username *: ubuntu
- Password *: (redacted)
- CPU count *: 1
- Time *: 30

At the bottom of the form is a button labeled "✓ Start Fault".

FIT CPU

Using the GUI is a straightforward process of selecting the desired fault and providing the VM details. In the example below a high CPU usage fault is chosen, and the address, username and password of the VM is entered. The number of CPUs on the machine is entered and then the amount of time to overload the CPU ~~for~~ in this case 30 seconds, as shown in Figure 4.

The GUI successfully complements the objectives achieved by the command line tool. It makes the fault injection tool highly accessible, allowing anyone to find vulnerabilities and test the resiliency of their systems.

Installation

The source code can be found in the DICE GitHub repository. The repository contains the source code and a WAR file so it can be deployed on a server, such as Apache Tomcat. Once the image is deployed on the server it will be immediately available and ready to use.

Open Challenges

Future work will involve a more detailed classification of faults in conjunction with an analysis of the cause, effect and response of various fault scenarios. From a design and operational perspective an investigation will be undertaken to decide on how best to integrate with related services. There is a requirement for robust monitoring, graphing and analysis to operate in conjunction with the FIT. This could mean logically packaging a single solution, using a suite of tools or building new inherent features. This will result in firstly a wider scope of technologies that can be stressed with workloads in a similar way to MongoDB such as Storm, Spark and Cassandra. Further control over the timing, duration of faults will be investigated.

Application domain

Integration

The screenshot shows the DICE Fault Injection Tool's user interface. At the top, there is a logo for 'DICE Fault Injection Tool'. Below the logo, a header bar says 'Create faults on a DICE deployment within FCO'. The main area has a form for entering a 'Token' and an 'ID'. Below the form are three buttons: 'Start Fault', 'List Deployments', and 'List Nodes'. Underneath these buttons is a section for 'SSH key and JSON upload options' with two sub-options: 'Upload SSH key' and 'Upload inputs from a JSON file'. Each option has its own 'Choose' and 'Submit' buttons. The bottom section is labeled 'Output' and contains two large text areas: 'Completed Deployments' and 'Deployment Details', both of which are currently empty.

FIT Deployment

A major step forward in the development of the Fault Injection Tool is the integration with other DICE tools. This work incorporates the FIT deeper within the DICE toolset, and provides further useful functionality for users of the FIT. One tool in which the FIT has been fully integrated with is the DICE Deployment Service, developed by XLAB, which is shown in Figure 5. This new feature allows faults to be caused on all of the VMs which make up a deployment into a DICE virtual deployment container. The method in which this integration works is through the GUI version of the FIT. First, the user must acquire a token from the deployment service, in order to be able to authenticate with the API. From there, an option is given on the GUI to list all containers running on the DICE deployment service. The user can then choose the container they wish to cause faults on. A JSON file can also be uploaded in order to further customise the type of faults to be caused on certain VMs within the deployment. This is accomplished by matching a fault with the name of the component type that is associated (e.g., hosted on) with the VM in the application's deployment blueprint. After these attributes are provided, the desired faults are automatically caused on all of the selected VMs inside the container, to simulate the faults occurring at an application level. This enables that the user needs to fill in the form only once for a virtual deployment container, then use the same information for all the subsequent (re)deployments in the container. Additionally, we show in the next chapter the integration of FIT with the new dmon-gen utility to automate the generation of anomalies.

Additionally, there is the integration of FIT with the new dmon-gen utility to automate the generation of anomalies. Using this we can also specify FIT specific parameters for each experiment. For example we can use this to stress the CPU and memory of all of the hosts on a given platform or to use exactly 50% of the available resources.

Another area in which integration is being explored is the incorporation of the monitoring software developed by IeAT to the FIT GUI. If this is feasible, it will provide users with detailed statistics and analysis of how the VM(s) are performing before, during and after faults have been caused. This could provide a deeper understanding of the effects of the faults caused by the FIT application, and how the VMs/applications perform under the strain of these different faults being simulated.

The FIT can generate VM faults for use by application owners and VM admins. The tool is designed to run independently and externally to any target environment.

Validation

```
top - 08:33:21 up 22:27,  2 users,  load average: 0.07, 0.04, 0.05
Tasks: 67 total,  1 running, 66 sleeping,  0 stopped,  0 zombie
%Cpu(s): 0.7 us, 1.0 sy, 0.0 ni, 98.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 501772 total, 347048 used, 154724 free, 51092 buffers
KiB Swap: 0 total, 0 used, 0 free. 231424 cached Mem

PID USER      PR  NI    VIRT    RES   SHR S %CPU %MEM     TIME+ COMMAND
 1 root      20   0 33640  2896 1444 S 0.3 0.6  0:10.01 init
 69 root      20   0      0      0    0 S 0.3 0.0  0:43.09 kworker/0:2
```

Before CPU Overload

```
top - 08:36:28 up 22:30,  2 users,  load average: 0.22, 0.07, 0.06
Tasks: 73 total,  3 running, 70 sleeping,  0 stopped,  0 zombie
%Cpu(s):100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 501772 total, 350672 used, 151100 free, 51120 buffers
KiB Swap: 0 total, 0 used, 0 free. 231592 cached Mem

PID USER      PR  NI    VIRT    RES   SHR S %CPU %MEM     TIME+ COMMAND
14533 ubuntu  20   0 7304    96    0 R 99.2 0.0  0:12.57 stress
14417 ubuntu  20   0 23536  1500 1092 R 0.7 0.3  0:00.56 top
```

During CPU Overload

To validate the impact of FIT usage the common task manager program ‘top’ found in many Unix-like operating systems was used. Using the Linux ‘top’ command on the target VM the current state of its resources can then be seen. Regarding CPU saturation, before running the CPU overload fault, the %Cpu usage is measured. While running the CPU overload the %Cpu quickly rises to near 100%. The stress command issued by the FIT would typically account for around 99.2% of the available CPU capacity. Regarding memory saturation, the FIT ‘stressmem’ feature is called with designated parameters. The tool first connects via SSH to the VM and determined the OS version by checking the /etc/*-release for the version of the OS (Ubuntu in our case). It then looks for the memory stress tool suitable for Ubuntu, for example Memtester. If the tool is not found first the DICE FIT installs the tool along with dependencies. Finally, the FIT calls Memtester to saturate memory in the target node. Again, a standard monitoring tool such as ‘top’ (in the %MEM column) will show the 2GB (or whatever was specified in the memory size parameter of stressmem) RAM available to the VM being saturated. The use of standard measurement tools already bundled with the OS means it is then easy for users to ensure the injected fault is having the desired effect.

In the following example, using the Linux ‘top’ command on the target VM the current state of its resources can be seen. Before running the CPU overload fault, the %Cpu usage is at 0.7% as seen in Figure 6. While running the CPU overload the %Cpu quickly rises to 100%. In the processes below we can see that the stress command is using 99.2% of the CPU as shown in Figure 7.

Conclusion

The main advantages of the FIT in comparison to other solutions are its availability as an open source solution, a command line version that makes it easy to integrate with other tools, a GUI that makes it easy to use for the non expert user and well documented instructions. An important differentiator is the cloud agnostic nature of the tool and its ability for consistent use on multiple target environments. Further future work will focus on the difficulties and possibilities of extensibility by external users, investigating limitations, consider different topologies, operating systems and vendor agnostic cloud provider infrastructure as well as evaluating the overhead of operation. Containerised environments will also be considered as future FIT targets to help understand the effect on microservices when injecting faults to the underlying host as well as the integrity of the containerised deployment. In the longer term other target cloud APIs could be added to the FIT as well as investigating related academic work.

References

1. [Fault Injection Tool](https://github.com/dice-project/DICE-Fault-Injection-Tool/) (<https://github.com/dice-project/DICE-Fault-Injection-Tool/>)
 2. [S. Han, K. G. Shin, and H. A. Rosenberg, "DOCTR: An IntegrateD SOftware Fault InjeCTOn EnviRonment for Distributed Real-time Systems," presented at International Computer Performance and Dependability Symposium, Erlangen; Germany 1995.]
 3. [S. Dawson, F. Jahanian, and T. Mitton, "ORCHESTRA: A Probing and Fault Injection Environment for Testing Protocol Implementations," presented at International Computer Performance and Dependability Symposium, Urbana-Champaign, USA, 1996.]
 4. [xception](http://www.criticalsoftware.com/en/products/p/xception/) ([https://www.criticalsoftware.com/en/products/p/xception/](http://www.criticalsoftware.com/en/products/p/xception/))
 5. [GridFIT](http://web.archive.org/web/20071030063006/http://wiki.grid-fit.org:80/bin/view/GridFIT/) ([https://web.archive.org/web/20071030063006/http://wiki.grid-fit.org:80/bin/view/GridFIT/](http://web.archive.org/web/20071030063006/http://wiki.grid-fit.org:80/bin/view/GridFIT/))
 6. [LFI](http://lfi.epfl.ch/) (<http://lfi.epfl.ch/>)
 7. [Memtester](http://pyropus.ca/software/memtester/) (<http://pyropus.ca/software/memtester/>)
 8. [Apache 2.0](http://www.apache.org/licenses/LICENSE2.0/) ([https://www.apache.org/licenses/LICENSE2.0/](http://www.apache.org/licenses/LICENSE2.0/))
-

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Fault_Injection&oldid=3369216'

This page was last edited on 29 January 2018, at 12:08.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Monitoring

Contents

- Introduction**
- Motivations**
- Existing solutions**
- How the tool works**
- Open Challenges**
- Application domains**
- Conclusion**
- References**

Introduction

Big Data technologies have become an ever more present topic in both academia and industrial world. These technologies enable businesses to extract valuable insight from their available data, hence more and more SMEs are showing increasing interest in using these types of technologies. Distributed frameworks for processing large amounts of data, such as Apache Hadoop^[1], Spark^[2], or Storm^[3] gained in popularity and applications developed on top of them are more and more prevalent. However, developing software that meets these high-quality standards expected for business-critical Cloud applications remains a challenge for SMEs. In this case model-driven development (MDD) paradigm and popular standards such as UML, MARTE, TOSCA^[4] hold strong promises to tackle this challenge. During development of Big Data applications it is important to monitor performance for each version of the application. Information obtained can be used by software architects and developers to track the evolution in time of the developed application. Monitoring is also useful in determining main factors that impact the quality of different application versions. Throughout the development stage, running applications tend to be more verbose in terms of logged information so that developers can get insights about the developed application. Due to verbosity of logs, data- intensive applications produce large amounts of monitoring data, which in turn need to be collected, pre-processed, stored and made available for high-level queries and visualization. It is clear that there is a need for a scalable, highly available and easy deployable platform for monitoring multiple Big Data frameworks. Which can collect resource-level metrics, such as CPU, memory, disk or network, together with framework level metrics collected from Apache HDFS, YARN, Spark and Storm.

Motivations

In all development environments and in particular in a DevOps one infrastructure and application performance monitoring is very important. It allows developers to check the current performance and to evaluate potential performance bottlenecks. Most monitoring tools and services focus on providing a discrete view of a particular application and/or platform. Meaning that if a DIA is based on more than one platform it is very likely that one monitoring solution will not support all of the platforms. DMon is designed in such a way that it is able to monitor the most common Big Data platforms as well as system/infrastructure metrics. This way during development only one monitoring solution is able to handle all monitoring needs.

Of course platform monitoring is not enough, most of the time developers require information about their application performance. DMon can be easily extended to collect metrics via [JMX](#) or even custom log formats. All of the resulting metrics can then be queried and visualize using DMon.

Existing solutions

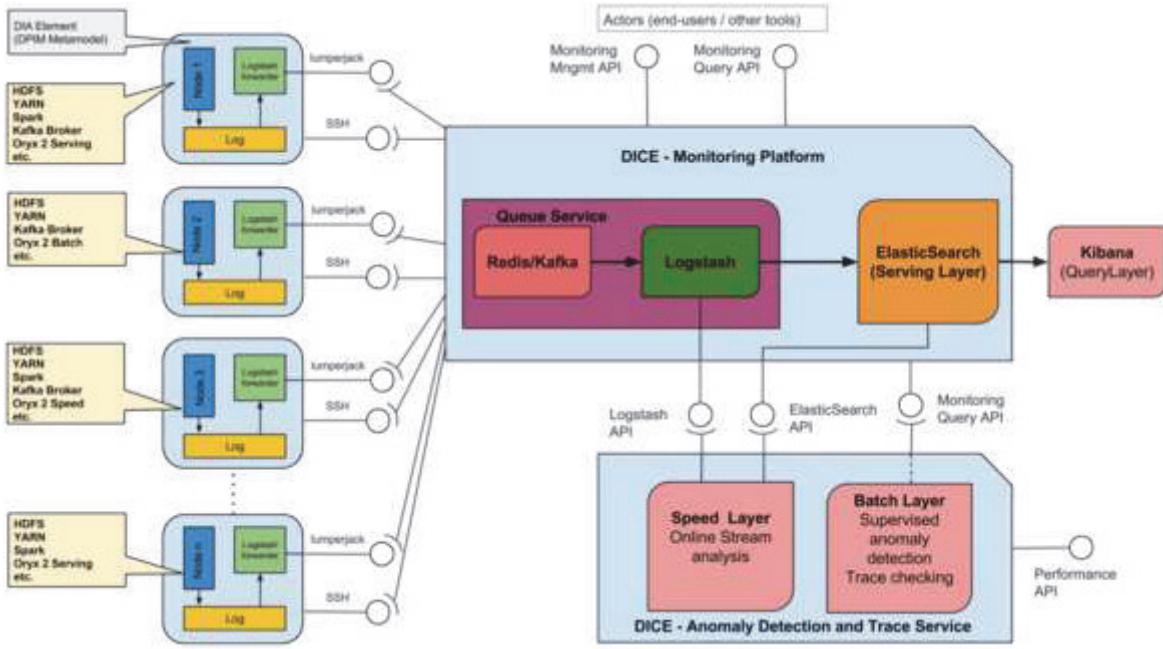
Currently on the market there are a lot of tools and services geared towards monitoring different cloud deployed resources and services. Big Data services are no different, here are just a few solutions which are used:

- Hadoop Performance Monitoring ^[5] provides an Hadoop inbuilt solution for quickly finding performance bottlenecks and provide a visual representation of the configuration parameters which might be tuned for better performance. Basically it is a lightweight monitoring UI for Hadoop serveOne of its main advantage is the availability in the Hadoop distribution and the ease of usage. On the other hand it proves to be fairly limited with regard to performance. For example, the time spent in gc by each of the tasks is fairly high.
- SequenceIQ^[6] provides a solution for monitoring Hadoop clusters. The architecture proposed by SequenceIQ and used in order to do monitoring is based on Elasticsearch^[7], Kibana and Logstash. The architecture has as its main objective obtaining a clear separation between monitoring tools and the existing Hadoop deployment. For achieving this three Docker containers are used. In a nutshell the monitoring solution consist of client and server containers. The server container takes care of actual monitoring tools. In this particular deployment Kibana is used for visualization and Elasticsearch for consolidation of the monitoring metrics. Through the capabilities of Elasticsearch one can horizontally scale and cluster multiple monitoring components. The client container contains the actual deployment of the tools that have to be monitored. This instance contains Logstash, Hadoop and the collectd modules. Logstash connects to the Elasticsearch cluster as a client and stores the processed and transformed metrics data there. Basically this solution consists of a collection of tools that are used in order to monitor different metrics from different layers. Among the main advantages of this solution one notices the ease of adding and removing different components from the system. Another interesting aspect of this architecture is the ease with which one can extract different informations from tools.
- Datastax^[8] provide a solution, OpsCenter , that can be integrated in order to monitor Cassandra^[9] installation. Using OpsCenter one can monitor different parameters of a Cassandra instance. Also it can handle many other parameters provided by the actual machines on which it runs. OpsCenter exposes an interactive web UI that allow administrator to add or remove nodes from the deployment. An interesting feature provided by the OpsCenter is the automatic load balancing. For integration of OpsCenter with other tools and services a developer API is provided.

How the tool works

The DICE Monitoring platforms (DMon)^[10] architecture is designed as a web service that enables the deployment and management of several subcomponents which in turn enable the monitoring of big data applications and frameworks. In contrast to other monitoring solutions, DMon aims to provide as much data as possible about the current status of the big data framework subcomponents. This intent brings a wide array of technical challenges, not present in more traditional monitoring solutions, as serving near real-time fine grained metrics entails a system that should exhibit a high availability, as well as easy scalability. Traditionally, web services have been built using a monolithic architecture where almost all components of a system ran in a single process (traditionally JVM). This type of architecture has some key advantages such as: deployment and networking are trivial while scaling such a system requires running several instances of the service behind a load-balancer instance. On the other hand, there are some severe limitations to this type of monolithic architecture, which would directly impacts the development of DMon. Firstly, changes to one component can have an unforeseen impact on seemingly unrelated areas of the application, thus adding new features or any new development can be potentially costly both in time and resources. Secondly, individual components cannot be deployed independently. This means that if one only needs a particular functionality of the service this cannot be decoupled and deployed separately thus hindering reusability. Lastly, even if components are designed to be reusable these tend to focus more on readability than on performance.

Considering these limitations of a monolithic architecture, we decided to use the so called microservice architecture for DMon, which is widely used in large Internet companies. This architecture replaces the monolithic service with a distributed system of lightweight services, which are by design independent and narrowly focused. These can be deployed, upgraded and scaled individually. As these microservices are loosely coupled, it better enables code reusability, while changes made to a single service should not require changes in a different service. Integration and communication should be done using HTTP (REST API) or RPC requests. We also want to group related behaviours into separate services. This will yield a high cohesion, which enables us to modify the overall system behaviour by only modifying or updating one service instead of many. DMon uses REST APIs for communication between different services with request payload encoded as JSON message. This makes the creation of synchronous or asynchronous messages much easier



DMon Architecture

The figure shows the overall architecture of the DMon platform, which will be part of a lambda architecture together with the anomaly detection and trace checking tools. In order to create a viable lambda architecture we need to create 3 layer: speed, batch and serving layers. Elasticsearch will represent the serving layer responsible for loading batch views of the collected monitoring data and enabling other tools/layers to do random reads on it. The speed layer will be used to look at recent data and represent it in a query function. In the case of anomaly detection this will mean using unsupervised learning techniques or using pre-trained models from the batch layer. The batch layer needs to compute arbitrary functions on large sections of the dataset stored in Elasticsearch. This means running long running jobs to train predictive models that can be instantiated on the speed layer. All trained models will then be stored inside the serving layer and accessed via DMon queries. The core components of the platform are Elasticsearch, for storing and indexing of collected data, and Logstash for gathering and processing logfile data. Kibana server provides a user-friendly graphical user interface. The main services composing DMon are the following: dmon-controller, dmon-agent, dmon-shipper, dmon-indexer, dmon-wui and dmon-mas. These services will be used to control both the core and node-level components.

Open Challenges

Big Data technologies are continuously evolving. Because of this any monitoring solution has to keep up with the ever changing metrics and metric exposing systems. DMon for example is capable of monitoring the most common platforms currently in use however new platforms such as Apache Flink are not yet supported.

Furthermore some basic data preprocessing such as averaging, windowing and filtering are already supported by DMon additional operations should could be required in future versions. This is also true for data exporting formats.

Application domains

DMon has been tested on all Big Data platforms currently supported in DICE. These include:

- Apache Yarn (including HDFS)
- Apache Spark (versions 1.6.x and 2.x.x)
- Apache Storm
- Cassandra
- MongoDB

In addition to these platforms DMon is also able to collectd via collectd a wide array of system metrics. In short it supports all collectd supported metrics. During development application metrics are in some ways more important than the platform metrics, because of this DMon can be easily extended to support any log or metrics format required via Logstash filter plugins.

An important point to mention is that the metrics collected for a DIA that uses more than one of the supported platform can be easily aggregated and exported based on the timestamp they were received. For example if we have a DIA that uses Spark which in turn runs on top of Yarn and HDFS, DMon is able to show all the metrics for any given timeframe for the aforementioned platforms.

Conclusion

The DICE Monitoring Platform is a distributed, highly available system for monitoring Big Data technologies, as well as system metrics. Aligning DMon objectives to DICE visions, that is bringing together Model-Driven Development and DevOps to enable fast development of high-quality data intensive applications.

DMon features automation at numerous levels: deployment of software components the nodes of monitored cluster, easy management of the monitoring platform itself, or automatic creation of visualizations based on collected data. Thanks to close integration with DICE Deployment Service (based on Cloudify and Chef cookbooks), software engineers/architects only need to annotate appropriate nodes in the DDSM model or TOSCA blueprint as monitorable and the Deployment service will install and configure the agents on selected nodes, so that the moment the DIA is deployed on the cluster the runtime data will be flowing into the DMon platform, with absolutely no manual intervention from end-users.

Engineered using a micro-services architecture, the platform is easy to deploy, and operate, on heterogeneous distributed Cloud environments. We reported successful deployment on Flexiant Cloud Orchestrator and OpenStack using Vagrant scripts.

The work done on this tool has highlighted the need for a specialized monitoring solution in a DevOps environment. The usage of a lightweight yet high throughput distributed monitoring solution capable of collecting thousands of metrics across a wide array of Big Data services is of paramount importance. This importance is even more evident when dealing with preliminary versions of applications that are in development. Monitoring solutions like DMon can provide an excellent overview of the current performance of an application version.

References

1. <http://hadoop.apache.org/>
 2. <https://spark.apache.org/>
 3. <http://storm.apache.org/>
 4. <https://www.oasis-open.org/committees/tosca>
 5. <https://www.datadoghq.com/blog/monitor-hadoop-metrics/>
 6. <http://sequenceiq.com/>
 7. <https://www.elastic.co/>
 8. <https://www.datastax.com/>
 9. <http://cassandra.apache.org/>
 10. <https://github.com/dice-project/DICE-Monitoring>
-

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Monitoring&oldid=3338180

This page was last edited on 4 December 2017, at 12:22.

Text is available under the [Creative Commons Attribution-ShareAlike License](#), additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Anomaly Detection

Contents

- Introduction**
- Motivations**
- Existing solutions**
- How the tool works**
- Open Challenges**
- Application domains**
- Conclusion**
- References**

Introduction

In anomaly detection the nature of the data is a key issue. There can be different types of data such as: binary, categorical or continuous. In DICE we deal mostly with the continuous data type although categorical or even binary values could be present. Most metrics data relate to computational resource consumption, execution time etc. There can be instances of categorical data that denotes the status/state of a certain job or even binary data in the form of boolean values. This makes the creation of data sets on which to run anomaly detection an extremely crucial aspect of ADT^[1], because some anomaly detection methods don't work on categorical or binary attributes.

It is important to note that most, if not all, anomaly detection techniques and tools, deal with point data, meaning that no relationship is assumed between data instances. In some instances this assumption is not valid as there can be spatial, temporal or even sequential relationships between data instances. This in fact is the assumption we are basing ADT on with regard to the DICE context.

All data in which the anomaly detection techniques will use are queried from monitoring platform (DMon). This means that some basic statistical operations (such as aggregations, median etc.) can already be integrated into the DMon query. In some instances this can reduce the dataset size in which to run anomaly detection.

An extremely important aspect of detecting anomalies in any problem domain is the definition of the anomaly types that can be handled by the proposed method or tool. In the next paragraphs, we will give a short definition of the classification of anomalies in relation to the DICE context.

First, we have point anomalies which are the simplest types of anomalies, represented by data instances that can be considered anomalous with respect to the rest of the data. Because this type of anomaly is simple to define and check, a big part of research effort will be directed towards finding them. Our intend was to investigate these types of anomalies and included them in DICE ADT. However, as there are a lot of existing solutions already on the market this is not be the main focus of ADT instead we use the Watcher^[2] solution from the ELK stack to detect point anomalies. A more interesting type of anomalies in relation with DICE are the so called contextual anomalies. These are considered anomalous only in a certain context and not otherwise. The context is a result of the structure from the data set. Thus, it has to be specified as part of the problem formulation.

When defining the context, we consider contextual attributes which are represented by the neighbours of each instance and behavioural attributes which describe the value itself. In short, anomalous behaviour is determined using the values for the behaviour attributes from within the specified context.

The last types of anomalies are called collective anomalies. These anomalies can occur when a collection of related data instances are anomalous with respect to the entire data set. In other words, individual data instances are not anomalous by themselves. Typically collective anomalies are related to sequence data and can only occur if data instances are related.

Motivations

During the development phases of an application, performance bottlenecks as well as unwanted or undocumented behaviour is commonplace. A simple monitoring solution is not a viable solution as the metrics usually require expert knowledge of the underlying platform architecture. This is especially true in DevOps environments. The DICE anomaly detection tool contains both supervised and unsupervised methods which are able to flag undesired application behaviours.

The unsupervised methods are useful during the initial stages of a newly developed application. In this situation developers have a hard time of deciding if a particular performance profile is normal or not for the application. Supervised methods on the other hand can be used by developers in order to detect contextual anomalies between application versions. In contrast to the unsupervised methods, this requires a training set so that they can be trained to detect different anomalous instances.

Existing solutions

There are a wide range of anomaly detection methods currently in use. These can be split up into two distinct categories based on how they are trained. First there are the methods used in supervised methods. In essence these can be considered as classification problems in which the goal is to train a categorical classifier that is able to output a hypothesis about the anomaly of any given data instances. These classifiers can be trained to distinguish between normal and anomalous data instances in a given feature space. These methods do not make assumptions about the generative distribution of the event data, they are purely data driven. Because of this, the quality of the data is extremely important.

For supervised learning methods, labelled anomalies from application data instances are a prerequisite. False positives frequency is high in some instances. This can be mitigated by comprehensive validation/testing. Computational complexity of validation and testing can be substantial and represents a significant challenge, which has been taken into consideration in the ADT tool. Methods used for supervised anomaly detection include but are not limited to: Neural Networks, Neural Trees, ART^[3], Radial Basis Function, SVM, Association Rules and Deep Learning based techniques. In unsupervised anomaly detection methods, the base assumption is that normal data instances are grouped in a cluster in the data while anomalies don't belong to any cluster. This assumption is used in most clustering based methods, such as: DBSCAN^[4], ROCK, SNN FindOut, WaveCluster. The second assumption on which K-Means, SOM, Expectation Maximization (EM) algorithms are based is that normal data instances belong to large and dense clusters while anomalies in small and sparse ones. It is easy to see that the effectiveness of each of unsupervised, or clustering based, methods is largely based in the effectiveness of individual algorithms in capturing the structure of normal data instances.

It is important to note that these types of methods are not designed with anomaly detection in mind. The detection of anomalies is more often than not a product of clustering based techniques. Also, the computational complexity in the case of clustering based techniques can be a serious issue and careful selection of the distance measure used is a key factor.

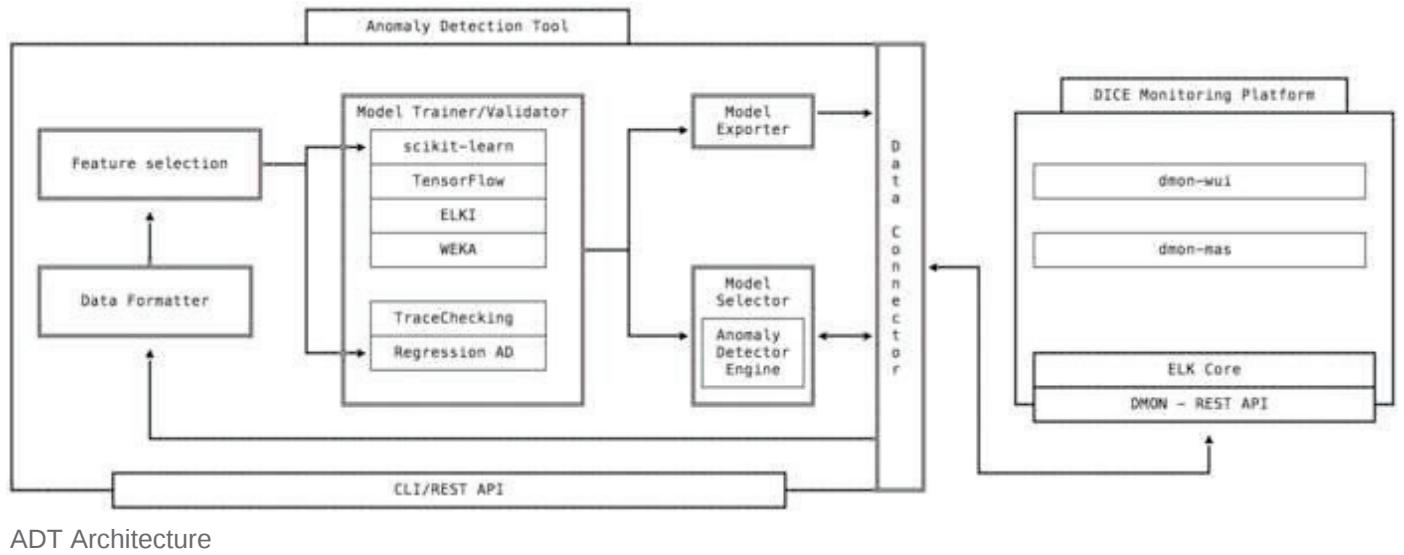
How the tool works

The ADT is made up of a series of interconnected components that are controlled from a simple command line interface. This interface is meant to be used only for the initial version of the tool. Future versions will feature a more user friendly interface. The full architecture can be viewed in the architecture figure from this section. In total, there are 8 components that make up ADT. The general architecture is meant to encompass each of the main functionalities and requirements identified during the requirements deliverables^[5].

First, we have the data-connector component, which is used to connect to DMon. It is able to query the monitoring platform and also send it new data. This data can be detected anomalies or learned models. For each of these types of data, data-connector creates a different index inside DMon. For anomalies, it creates an index of the form anomaly-UTC, where UTC stands for Unix time, similarly to how the monitoring platform deals with metrics and their indices. This means that the index is rotated every 24 hours.

After the monitoring platform is queried, the resulting dataset can be in JSON, CSV or RDF/XML. However, in some situations, some additional formatting is required. This is done by the data formatter component. It is able to normalize the data, filter different features from the dataset or even window the data. The type of formatting the dataset may or may not need is highly dependant on the anomaly detection method used. The feature selection component is used to reduce the dimensionality of the dataset.

Not all features of a dataset may be needed to train a predictive model for anomaly detection. So in some situations, it is important to have a mechanism that allows the selection of only the features that have a significant impact on the performance of the anomaly detection methods. Currently, only two types of feature selection is supported. The first is Principal Component Analysis (from Weka) and Wrapper Methods.



The next two components are used for training and then validating predictive models for anomaly detection. For training, a user must first select the type of method desired. The dataset is then split up into training and validation subsets and later used for cross validation. The ratio of validation to training size can be set during this phase. Parameters related to each method can also be set in this component. Validation is handled by a specialized component which minimizes the risk of overfitting the model as well as ensuring that out of sample performance is adequate. It does this by using cross validation and comparing the performance of the current model with past ones.

Once validation is complete, the model exporter component transforms the current model into a serialized loadable form. We use the PMML^[6] format wherever possible in order to ensure compatibility with as many machine learning frameworks as possible. This makes the use of ADT in a production like environment much easier. Not all model are currently exportable in this format, in particular neural network based models are not compatible. The resulting model can be fed into DMon. In fact, the core services from DMon (specifically Elasticsearch) have to role of a serving layer from a lambda architecture. Both detected anomalies and trained models are stored in the DMon and can be queried directly from the monitoring platform. In essence, this means that other tools from the DICE toolchain need to know only the DMon endpoint in order to see what anomalies have been detected.

Furthermore, the training and validation scenarios is in fact the batch layer while unsupervised methods and/or loaded predictive models are the speed layer. Both these scenarios can be accomplished by ADT. This integration is an open-challenge detailed in the next section.

The last component is the anomaly detection engine. It is responsible for detecting anomalies. It is important to note that it is able to detect anomalies however it is unable to communicate them to the serving layer (i.e. DMon). It uses the dmon-connector component to accomplish this. The anomaly detection engine is also able to handle unsupervised learning methods. We can see this from the architecture figure that the Anomaly detection engine is in some ways a subcomponent of the model selector which selects both pre-trained predictive models and unsupervised methods.

Open Challenges

Currently, the anomaly detection tool relies on state of the art techniques for classification and anomaly detection. However, as this field is constantly evolving, the integration of new algorithms will be required. This integration is made easy by the internal architecture of the tool. Furthermore, several methods dealing with extremely unbalanced datasets have still to be explored. Some of the future improvements/challenges are:

- Inclusion of Oversampling and Undersampling techniques such as SMOTE^[17] and ADASYN
- Distributed hyperparameter optimization (i.e. running on Spark)
- Bayesian hyperparameter optimization
- Stacking and Bagging meta learning algorithms/methods
- Usage of deep learning techniques
 - Currently, we use the Keras library with a Tensorflow backend for training Neural Networks. However, no deep learning topologies have been tested yet.

Application domains

Because of the close integration with the monitoring platform ([DMon](#)) the anomaly detection tool can be applied to any platforms and applications supported by it. For a Storm based DIA, the anomaly detection tool queries DMon for all performance metrics. These metrics can be queried per deployed Storm topology. The workflow for utilising the tool is as follows:

- Query the data
 - Aggregate different data sources (for example system metrics with Storm metrics)
 - Filter the data
- Specify the anomaly detection method to use
 - Set training or detection mode (pre-trained model has to be selected by user)
 - For supervised methods a training data set has to be supplied by the user (including target values)
 - If no target value is provided, ADT considers the last column as target.

The resulting anomalies are stored inside DMon in a separate index. Consequently, anomalies can be queried and visualised like any other metric inside DMon. As mentioned before, because of the tight integration with DMon and the metric agnostic nature of ADT, applying it to other tools supported by DMon and DICE requires the same steps as for Storm.

It is also worth mentioning that ADT constructs the training and testing sets based on the structure from DMon. This means that as long as DMon has a flat representation of the metrics, ADT can use them.

Conclusion

The Anomaly detection tool developed during DICE is able to use both supervised and unsupervised methods. In conjunction with the DMon monitoring platform, it forms a lambda architecture that is able to both detect potential anomalies as well as continuously train new predictive models (both classifiers and clusterers). It is able to successfully detect performance related anomalies, thanks in part to the integrated preprocessing, training and validation modules.

It is also important to point out that the anomaly detection tool is in essence technology agnostic. It doesn't require apriori knowledge about the underlying technologies. The only context that it is given, is the one which is inherent in the structure of the training/validation dataset. This makes it ideal for novice users in developing DIAs.

References

1. <https://github.com/dice-project/DICE-Anomaly-Detection-dot>
 2. <https://www.elastic.co/guide/en/watcher/current/actions.html>
 3. <http://cns.bu.edu/Profiles/Grossberg/CarGro2003HBTNN2.pdf>
 4. <http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html>
 5. <http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2016/05/Requirement-Specification-M16.pdf>
 6. <http://dmg.org/pmmi/v4-1/GeneralStructure.html>
 7. <https://www.jair.org/media/953/live-953-2037-jair.pdf>
-

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Anomaly_Detection&oldid=3348361'

This page was last edited on 15 December 2017, at 13:14.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Trace Checking

Contents

Introduction

Motivations

Existing solutions

How the tool works

The architecture

The trace-checking process

Open challenges

Application domains

A test case - Wikistats

Conclusion

Introduction

Trace checking is an approach for the analysis of system executions that are recorded as sequences of timestamped events. Collected logs are analyzed to establish whether the system logs satisfy a property, usually specified in a logical language. In the positive case, the sampled system behavior conforms with the constraints modeled by the property; conversely, the system behavior simply does not satisfy the property.

When the language that is used to specify properties allows for the usage of temporal operators, trace checking is a way to check the correctness of the ordering of the events occurring in the system and of the time delays between pairs of events. For instance, if a property requires that *all the emitting events of a certain node occur not more than ten millisecond after the latest receive event*, then checking the property over a trace results in a boolean outcome, which is positive if the distance between two consecutive and ordered pair of emit and receive events is less than ten milliseconds.

The tool that is available in DICE to perform trace-checking (of Storm applications) **DICE-TraCT**.

Motivations

Trace checking is especially useful when the aggregated data that are available from the monitoring system are not enough to conclude the correctness of the system executions with respect to some specific criteria. In some cases, in fact, these criteria are application dependent as they are related to some non-functional property of the application itself and they do not depend on the physical infrastructure where the application is executed.

Trace checking is a possible technique to achieve this goal and can be used on purpose to extract information from the executions of a running application.

Logical languages involved in the trace checking analysis are usually extensions of metric temporal logics which offer special operators called aggregating modalities. These operators hold if the trace satisfies particular quantitative features like, for instance, a specific counting property of events in the trace.

According to the DICE vision, trace checking is performed after verification to allow for continuous model refinement. The result obtained through the log analysis confirms or refutes the outcome of the verification task, which is run at design time. The value of the parameters in the design-time model is compared with the value at runtime; if both are “compatible” then the results of verification are valid, otherwise the model must be refined.

Existing solutions

The research area related to runtime verification is very active on trace-checking. Various are the tools that support trace-checking and that provide either on-line or off-line analysis. Trace-checking in DICE is applied off-line but further extensions to the framework might consider also on-line analysis to improve the model-driven refinement that designers can perform. As far as the team knowledge, DICE is (one of) the first attempt which uses trace-checking analysis for supporting model-driven development of DIA. Trace-checking engines are tools that can be used in the assessment process and the choice of a specific one depends on the kind of analysis that designers wants to achieve. The International Competition on Runtime Verification is the best reference to compare available tools and techniques. Some of the tools that were involved in the competition and that might be considered as alternative to the ones used and implemented in DICE are the following:

- Rithm2 (paper): efficient parallel algorithm for verifying log traces with an extended version of ITL with counting semantics.
- MonPoly: first-order extension of ITL enriched with aggregating modalities.
- ARTiMon: a tool for the analyses of flow of timestamped observations that can be used to detect hazards expressed in a temporal logic-based language.

How the tool works

The outcome of the trace-checking analysis is used for refining the model of the application at design time that is verified with D-VerT.

Trace-checking is performed at run-time but it is based on annotated DTSM models, defined at design-time, that contain the topology undergoing the analysis.

DICE-TraCT has been designed to encompass Soloist language Soloist which offers the following class of aggregating modalities:

- number of occurrences of an evente in a time window of length d ,
- maximum/average number of occurrences of an evente, aggregated over right-aligned adjacent non-overlapping subintervals (of size h) in a time window of length d ,
- average time elapsed between a pair of specific adjacent and alternating events and e occurring in a time window of length d .

However, some basic functionalities, such as the averaging of event in a given time window, have been developed in a simple trace analyser to improve the flexibility of the tool. The user can therefore exploit the most appropriate engine to perform the log analysis.

The architecture

DICE-TraCT has a client-server architecture: the client component is an Eclipse plugin that is fully integrated with the DICE IDE and the server component is a RESTful web server. The client component manages the transformation from the DTSM model defined by the user to an intermediate JSON object which is then used to invoke the server. The server component, based on the content of the JSON file generates the trace-checking instance for the trace-checking engine.

DICE-TraCT is an Eclipse plugin that can be executed within the DICE IDE. To carry out the trace-checking analysis, Storm topologies have to be specified by means of UML class diagrams and activity diagrams with the DICE::Storm profile. The diagrams can easily be drawn by dragging and dropping Eclipse items that are available in the DICE toolbars whereas the annotations can be specified in the bottom panel of the DICE IDE.

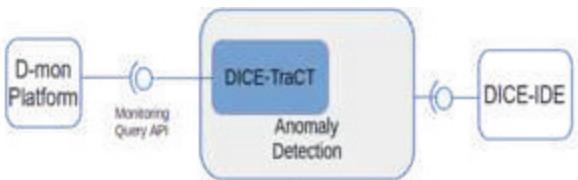


Figure 1. The architectural connectors involving DICE-TraCT

DICE-TraCT can be set through a suitable Run Configuration window allowing users to launch the analysis. The user can specify:

- the file containing the model undergoing the analysis,
- the time-bound limiting the duration of the time window that is analyzed,
- the set of Storm bolts and spouts whose metrics (*sigma* and *avg_emit_rate*) will be calculated by means of the topology logs,
- the IP address and the port where the **DICE-TraCT** server is running and
- the IP address and the port of the monitoring platform.

The trace-checking process

Trace-checking is performed on a running deployed application that has been designed by means of an abstract model verified by **D-VerT** (verification tool), and possibly with the results obtained from the simulation and optimization tools. The parameter values that were employed to carry out the analysis with **D-VerT** are compared with the current values obtained by the analysis of the application log traces performed by **DICE-TraCT**. Based on the kind of the analysis (counting, average or quantitative) and the structure of the topology, **DICE-TraCT** retrieves the logs from the monitoring platform (D-Mon) and performs the analysis. When the outcome is available, the result is shown in the DICE IDE to allow the user to improve the model of the application, rerun the verification tool and, based on the outcome of the analysis, change the implementation if needed. The process is iterative and can be repeated until the verification tool does not detect anomalies that might lead the deployed application to undesired behaviours at runtime.

Open challenges

Trace-checking of application logs can be achieved successfully when logs contain suitable entries that enable the evaluation of temporal formulae or metrics. When the property to be checked only refers to events of the application that are tracked by the monitoring service of the data-intensive platform which runs the application, trace-checking can be employed simply by defining the property (or metric) to assess and by running the trace-checker on a selected log history. On the other hand, when the property to be checked refers to application events that are not natively monitored by the framework, trace-checking requires an ad-hoc instrumentation of the application code that allows the monitoring service to log those application-specific events needed for the evaluation of the property of interest.

Application domains

DICE-TraCT currently supports Storm logs analysis.

The current implementation of **DICE-TraCT** enables the analysis of two distinct parameters which are tightly related to the verification analysis carried out by **D-VerT**. In particular, the two available metrics that **DICE-TraCT** can elaborate are:

- *avg_emit_rate* that is the emitting rate of a spout node, defined as the number of tuple emitted per second, and
- *sigma* that is the ratio between the number of tuples produced by a bolt node in output and the number of tuples that it has received in input.

The value of the metrics that is obtained by trace-checking helps designers in tuning the model used for verification with **D-VerT**.

The trace-checking analysis can be restricted only to a specific time window and consider only the log entries that occur in a limited amount of time. The size determines how many log events are considered to carry out the evaluation of the metrics (The unit measurement is the millisecond).

DICE-TraCT implements three different kind of analysis calculating the metrics. Each one depends on the features of the underlying engines that are used to analyse the logs. Logical languages involved in the trace checking analysis are usually extensions of metric temporal logics which offer special operators called aggregating modalities. These operators are useful to count events in the logs or calculate the average number of the occurrences over a certain time window and compare the value with a given threshold. Therefore, the outcome of the analysis is always a boolean answer that can be either yes or no. **DICE-TraCT** is based on Soloist but the current implementation of **DICE-TraCT** can also call a simpler trace-checker than the one for Soloist, that is developed to provide quantitative informations calculated from the logs.

The following kind of analysis are available:

- *counting*: it enables the evaluation of the selected metric by means of the so called Counting formulae of the Soloist logic. This choice will call the Soloists trace-checker.
 - *average*: it enables the evaluation of the selected metric by means of the so called Average formulae of the Soloist logic. This choice will call the Soloists trace-checker.
 - *quantitative*: it enables the quantitative analysis of the logs which extracts the value of the metric.

The analysis instantiated by the [Soloist trace checker](#) is designed to provide a boolean result that is calculated by comparing the number of occurrences of an event in the log with a user defined threshold. The relations $<$, $=$ and $>$ and the threshold value can be specified via an appropriate configuration window when **DICE-TraCT** is launched.

A test case - Wkistats

DICE-TraCT is an Eclipse plugin that can be executed within the DICE IDE. To carry out log analysis, Storm topologies have to be specified by means of UML class diagrams and activity diagrams. The diagrams can easily be drawn by dragging and dropping Eclipse items that are available in the DICE toolbars whereas the annotations can be specified in the bottom panel of the DICE IDE.

Figure 2 depicts the [Storm](#) topology implementing the [Wikistats](#) topology (see the Github [repository](#) for implementation details). The diagram is a DTSM model which is also used for the analysis with **D-VerT**.

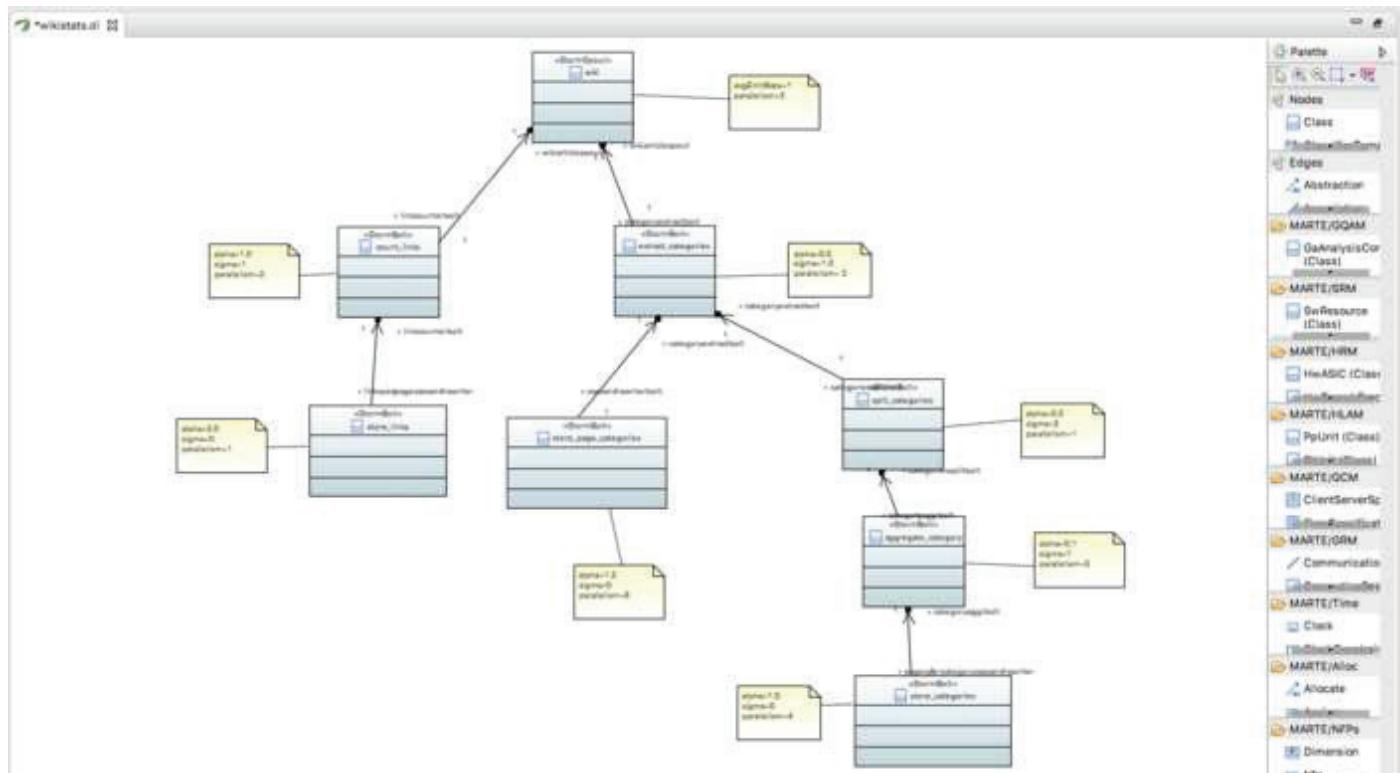


Figure 2. Wikistats topology

DICE-TraCT can be set through a Run Configuration window allowing users to launch the analysis. The user can specify:

- the file containing the model undergoing the analysis,
 - the Storm components undergoing analysis,

- the IP address and the port where the **DICE-TraCT** server is running.

The analysis allows for the extraction of model parameters related to verification, i.e., values that are used in the formal model analysed by **D-VerT**. DICE-TraCT extracts the following parameter values:

- sigma: ratio between number of tuples emitted by a bolt and number of tuples received by the bolt.
- alpha: average time required to elaborate a tuple by a bolt.
- emit rate: average emit rate of a spout.

Figure 3 shows the run configuration for **DICE-TraCT** and the list of the components defining the Wikistats topology. Moreover, the last column on the right-hand side of the figure shows the result of the analysis for some components of the topology after the execution of **DICE-TraCT**.

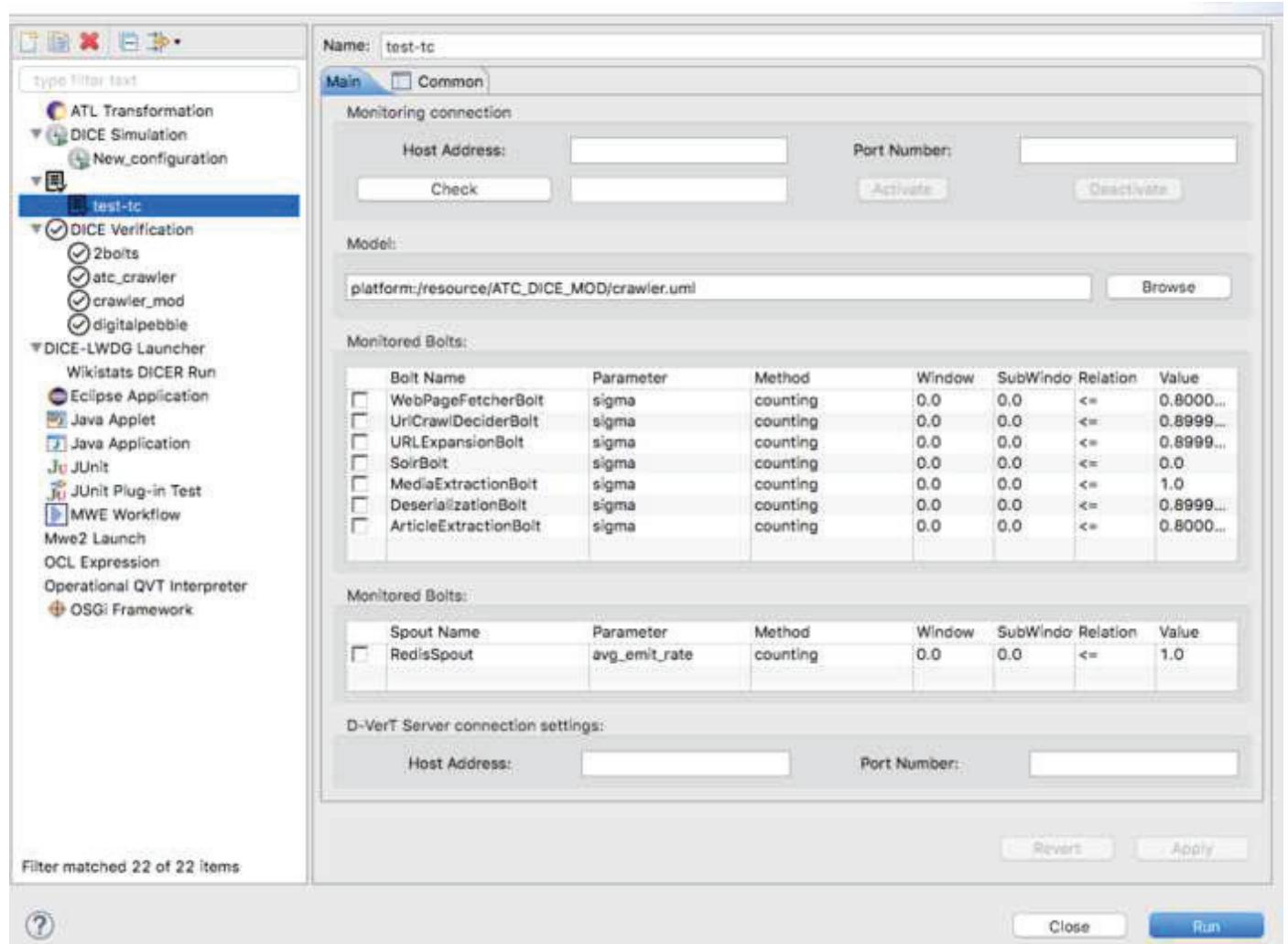


Figure 3. DICE-TraCT run configuration and results

Conclusion

DICE-TraCT has been developed to perform log analysis of Storm applications. It is used to assess the runtime behavior of a deployed application as log traces collected from the monitoring platform are analyzed to certify the adherence to the behavioral model that is assumed at design time. If the runtime behavior does not conform to the design, then the design must be refined and later verified to obtain a new certification of correctness. Trace-checking can be applied to supply information to the verification task carried out by **D-VerT**. Trace checking extracts from real executions the parameter values of the model used by **D-VerT** that are not available from the monitoring service of the framework, as they are inherently specific of the modeling adopted for the verification.

This page was last edited on 29 January 2018, at 09:01.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Iterative Enhancement

Contents

Introduction

Motivation

Existing Solutions

How the Tool Works

DICE-FG

DICE-APR

Open Challenges

Application domain: known uses

DICE FG

DICE-APR

Conclusion

References

Introduction

The goal of DICE is to offer a novel UML profile and tools that will help software designers reasoning about the quality of data-intensive applications, e.g., performance, reliability, safety and efficiency. Furthermore, DICE develops a new methodology that covers quality assessment, architecture enhancement, continuous testing and agile delivery, relying on principles of the emerging DevOps paradigm. In particular, one of the goals of DICE is to build tools and techniques to support the iterative improvement of quality characteristics in data-intensive applications obtained through feedback to the developers that will guide architectural design change.

To achieve that goal, DICE Enhancement tool is developed to provide feedback to DICE developers on the application behaviour at runtime, leveraging the monitoring data from the DICE Monitoring Platform (DMon)^[1], in order to help them iteratively enhance the application design. DICE Enhancement tool introduces a new methodology and a prototype to close the gap between measurements and UML diagrams. It correlates the monitoring data to the DICE UML models, with the aim of bridging the semantic gap between UML abstractions and concrete system implementation. Based on the acquired data, DICE Enhancement tool allows the developer to conduct within the DICE IDE more precise simulations and optimizations, that can rely on experimental data, rather than guesses of unknown parameters. DICE Enhancement tool also supports the developer in carrying out refactoring scenarios, with the aim of iteratively improving application quality in a DevOps fashion. According to our knowledge, no mature methodology appears available in the research literature in the context of data-intensive applications (DIAs) to address the difficult problem of going from measurements back to the software models, annotating UML to help to reason about the application design. DICE Enhancement tool aims at filling this gap.

The core components of the DICE Enhancement tool are two modules:

- DICE Filling-the-Gap (FG) module, a tool focusing on statistical estimation of UML parameters used in simulation^[2] and optimization tool^[3]. The tool provides the data to parameterize application design-time UML models by relying on the monitoring information collected at runtime. The goal is to enhance and automate the delivery of application performance information to the developer

- DICE Anti-Patterns & Refactoring (APR) module, a tool for anti-patterns detection and refactoring. The tool provides suggesting improvements to the designer of DIAs, based on observed and predicted performance and reliability metrics. The goal is to optimize a reference metric, such as maximize latency or minimize mean time to failure (MTTF).

Motivation

Motivated by the problem of inferring the bad practices in software design (i.e., performance anti-patterns) according to the data acquired at runtime during testing and operation, especially performance data, we developed DICE Enhancement tool. Since the abstraction levels between system runtime and design time are different, it is essential to the developer to obtain the runtime information, especially performance metrics, and reflect them into design-time model to reason on the quality of an application design and infer refactoring decisions.

To support performance analysis at the design time model, developers need to rely on software performance models for further analysis and evaluation. However, to provide reliable estimates, the input parameters must be continuously updated and accurately estimated. Accurate estimation is challenging because some parameters are not explicitly tracked by log files, requiring deep monitoring instrumentation that poses large overheads, unacceptable in production environments. Furthermore, performance Anti-Patterns (AP) are recurrent problems identified by incorrect software decisions. Software APs are largely studied in the industry. The increasing size and complexity of the software projects involve the rising of new obstacles more frequently. For that reason, the identification of AP at the early steps of the project life cycle saves money, time and effort. In order to detect the performance anti-patterns of DIAs, firstly the design-time model (i.e., architecture model) and performance model need to be specified, as well as the Model-to-Model transformation rules. Architecture model, as the system design time model, is specified by UML in DICE. In practice, developers typically use the activity diagram and deployment diagram of UML to modelling the system behaviour and infrastructure configuration. As the state of the art, UML is a general-purpose modelling language in the field of software engineering, and it provides a standard way to visualize the design of a system. Despite its popularity, UML is not suitable for automated analysis (e.g., performance evaluation). As a result, model analysis phases need to transform the annotated UML diagrams to performance models, e.g., Petri Net^[4], Layered Queueing Network (LQN)^[5]; for our work also considers the LQN as the performance model. As far as the performance metrics are concerned, LQN has advantages over UML, since LQN not only describes these metrics in a compact and understandable way but also supported by various analytical and simulation tools, e.g., LINE^[6], lqns^[7], and thus allows automating system performance analysis for further performance anti-patterns detection further AP detection

In order to achieve the above goal, we developed DICE enhancement tools to close the gap between runtime performance measurements and design time model for the anti-patterns detection and refactoring.

Existing Solutions

DICE-FG has been initially developed relying on a baseline, called FG, provided by the MODAClouds project as a way to close the gap between Development and Operations. Within DICE, the tool has undergone a major revision and is being integrated and adapted to operate on DIAs datasets. Architectural changes have been introduced in DICE-FG, compared to the original FG. Different from DICE-FG, APR has no baseline software to start from, since the only available tools in this space are not for UML. Hence it is an original contribution of DICE, to our knowledge novel in the UML space.

Throughout our desktop research, we have discovered the following solutions that can be considered as direct competitors to our DICE Enhancement Tool.

LibReDE^[8]: a library of ready-to-use implementations of approaches to resource demand estimation that can be used for online and offline analysis. LibReDE is used in general distributed systems while DICE-FG is designed for Data Intensive Applications; LibReDE mainly focuses on the resource demand estimation for performance model while DICE-FG concerns both performance and reliability estimation; the measurement data which is the input of the estimation for LibReDE is read from standard CSV files while the format of the input data for DICE-FG is more popular JSON file via DMon; LibReDE is not able to reflect the estimation results to the design time model while DICE-FG continuous parameterizing designing time models (UML model annotated with DICE Profiles) with estimated performance & reliability metric, that can inform developers on how to refactor the software architecture.

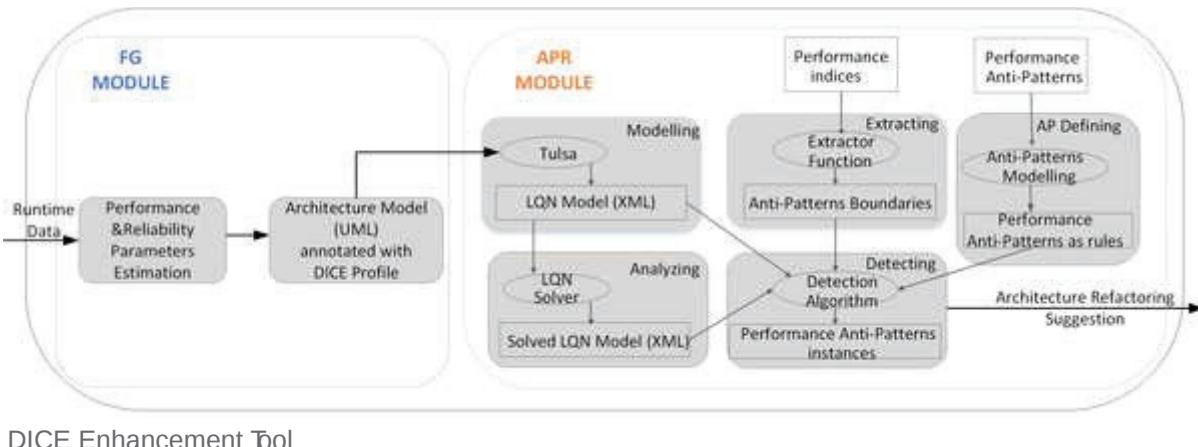
KieKer^[9]: is an extensible framework for monitoring and analyzing the runtime behavior of concurrent or distributed software systems. KieKer enables the application-level performance monitoring, including filters that allow the selection of data for further analysis. The DICE-FG, however, can reason on design-time models to deliver more accurate inferences of the model parameters from runtime monitoring data. Thus, rather than simply monitoring, the DICE-FG tool is envisioned as a machine-learning component that is aware of the application software architecture, and can use this to improve parameter learning.

PANDA^{[10][11]}: It is a framework for addressing the results interpretation and the feedback generation problems by means of performance anti-patterns. DICE-APR follows a similar methodology for automatically detecting and solving performance problems. The common thing between the PANDA and DICE-APR is they both leverage the UML model as their design time model (i.e., architecture model) while the input UML models for DICE-APR are annotated with the specific profiles (DPIM, DTSM and DDSM) which specifies the unique attributes for the Big Data application and platform. PANDA uses the Queueing Networking as its performance model while DICE-APR may consider Petri Net or Layered Queueing Networking model. DICE-APR's refactoring processing focuses on the Big Data application and will improve the former work on refactoring cloud-based applications, it will consider both the hardware and software knowledge of Big Data application.

PAD^[12]: PAD is a rule-based performance diagnosis tool, named Performance Antipattern Detection (PAD). PAD only focuses on Component Based Enterprise Systems, targeting EJB applications while DICE-APR concerns the Big Data applications and platform. They are both based on monitoring data from running systems while PAD's scope is restricted to the specific domain, whereas DICE-APR's starting point is the more general UML models of Data Intensive Application.

How the Tool Works

The DICE Enhancement tool is designed for iteratively enhancing the DIA quality. Enhancement tool aims at providing a performance and reliability analysis of Big Data applications, updating UML models with analysis results, and proposing a refactoring of the design if performance anti-patterns are detected. The following Figure shows the workflow for the Enhancement tool. It covers all of its intended functionalities which are discussed in details below



DICE Enhancement Tool

DICE-FG

As a core component of the Enhancement tool, the DICE-FG tool plays two roles:

- Updating parameters of design time model (UML models annotated with DICE Profile)
- Providing in the UML resource usage breakdown information for the data-intensive application

Together these features provide to the DICE designer the possibility to:

- Benefit from a semi-automated parameterization of simulation and optimization models. This supports the state goal of DICE of reducing the learning curve of the DICE platform for users with limited skills in performance and reliability engineering.
- Inspect in Eclipse the automated annotations placed by DICE-FG to understand the resource usage placed by a workload across software and infrastructure resources.

The main logical components of the DICE-FG tool are the Analyzer and the Actuator. Below we describe each component:

- **DICE-FG Analyzer.** The DICE-FG Analyzer executes the statistical methods necessary to obtain the estimates of the performance models parameters, relying on the monitoring information available on the input files.
- **DICE-FG Actuator.** The DICE-FG Actuator updates the parameters in the UML models, e.g., resource demands, think times, which are obtained from the DICE-FG Analyzer

DICE-APR

The DICE-APR module is designed to achieve the following objectives:

- Transforming UML diagrams annotated with DICE profile to performance model for performance analysis.
- Specifying the selected popular AP of DIAs in a formal way
- Detecting the potential AP from the performance model.
- Generating refactoring decisions to update the architecture model (manually or automatically) to fix the design flaws according to the AP solution.

The components of the APR module are Model-to-Model (M2M) Transformation (Tulsa), Anti-patterns Detection and Architecture Refactoring (APDR) as detailed below

- **Model-to-Model (M2M) Transformation (Tulsa):** The component provides the transformation of annotated UML model with DICE Profile into quality analysis model. The target performance model is Layered Queueing Networks.
- **Anti-patterns Detection and Refactoring (APDR)** The component relies on the analysis results of Tulsa. The selected anti-patterns (i.e., Infinite Wait (IW), Excessive Calculation (EC)) are formally specified for identifying if there are any anti-patterns issues in the model. According to the solution of discovered anti-patterns, refactoring decisions will be proposed, e.g., component replacement or component reassignment, to solve them. The Architecture model will be shared back to the DICE IDE for presentation, to the user in order to decide if the proposed modification should be applied or not.

Open Challenges

DICE Enhancement tool assumes that designer uses UML to represent the architecture model (i.e., activity diagram and deployment diagram) and use LQN model as the performance model. Without UML model, the user has to manually define the LQN model according to their architecture model. This may lead to the extra efforts. Besides, the DICE-APR currently can detect two performance anti-patterns and its taget are Storm-based Big Data applications.

Application domain: known uses

DICE FG

DICE-FG has been carried across a variety of technologies, including Cassandra^[13], Hadoop/MapReduce^[14]. For example, DICE-FG provides a novel estimator for **hostDemands**, which is able to efficiently account for all the state data monitored for a Big Data system. **hostDemands** of a Big Data application may be seen as the time that a request spends at a resource. For example, the execution time of a Cassandra query of type **c** at node **r** of a Cassandra cluster. A new demand estimation method called **EST-LE** (logistic expansion) has been included in the DICE-FG distribution. This method enables to use a probabilistic maximum-likelihood estimator for obtaining the **hostDemands**. Such approach is more expressive than the previous **est--qmle** method in that it includes information about the response time of the requests, in addition to the state samples obtained through monitoring. An obstacle that was overcome in order to offer this method is that the resulting maximum-likelihood method is computationally difficult to deal with, resulting in very slow execution times for the computation of the likelihood function. An asymptotic approximation is also developed that allows to efficiently compute the likelihood even in complex models with several resources, requests types, and high parallelism level.

DICE-APR

The practical use of the DICE-APR is for the Storm-based application. There are two reasons why the DICE-APR is suitable for Storm-based applications. First, since a Storm topology may be seen as a network of buffers and processing elements that exchange messages, it is thus quite natural to map them into a queueing network model. The interactions among the core elements (i.e., Spout and Bolt) of Storm applications and the deployment information can also be easily specified by the UML activity and deployment diagrams which is semantically similar to the LQN models. Thus, DICE-APR takes UML model (annotated with DICE and MARTE profile) of Storm-application as input and generates a performance model (i.e., Layered Queueing Networks (LQNs) model) for performance analysis. Second, in software engineering, APs are recurrent problems identified by incorrect software decisions at different hierarchical levels (architecture, development, or project management). Performance APs are largely studied in the industry. However, few of them focuses on the APs of data-intensive applications. Thus, we investigated the classic APs, Circuitous Treasure Hunt, Blob and Extensive Processing^[15] and define two Anti-Patterns (i.e., Infinite Wait and Excessive Calculation) of Storm-based applications for DICE-APR. The following are the problem statements of those APs and the corresponding solutions.

- Infinite Wait (IW): Occurs when a component must ask services from several servers to complete the task. If a large amount of time is required for each service, performance will suffer. To solve this problem, DICE-APR reports the component which causes the IW and provides component replication or redesign suggestions to the developer
- Excessive Calculation (EC): Occurs when a processor performs all of the work of an application or holds all of the application's data. Manifestation results in the excessive calculation that can degrade performance. To solve this problem, DICE-APR reports the processor which causes the EC and provides the suggestion, adding a new processor to migrate tasks, to the developer

Therefore, DICE-APR analyses the performance model of the Storm-based application by using LINE solver and provides refactoring suggestions if the above performance Anti-Patterns (APs) are detected.

Conclusion

The main achievements of DICE Enhancement tool are as follows:

DICE FG provide statistical estimation algorithms to infer resource consumption of an application and fitting algorithms to match monitoring data to parametric statistics distributions, and use the above algorithms to parameterize UML models annotated with the DICE profile.

DICE APR helps to transform the UML model annotated with DICE profile to LQN model, define and specify two APs and the corresponding AP boundaries for DIAs and detect the above APs from the models and provide the refactoring suggestions to guide the developer to update the architecture.

References

1. D4.2 Monitoring and Data Warehousing Tools - Final version, <http://www.dice-h2020.eu/resources/>
2. D3.4 DICE Simulation Tools - Final version, <http://www.dice-h2020.eu/resources/>
3. D3.9 DICE Optimization Tools - Final version <http://www.dice-h2020.eu/resources/>
4. Merseguer, J., Campos, J., Software performance modeling using uml and petri nets, Performance Tools and Applications to Networked Systems, 2965, 265-289(2004)
5. Altamimi, T., Zargari, M.H., Petriu, D., Performance analysis roundtrip: automatic Generation of performance models and results feedback using cross-model trace links, In: CASCON'16, Toronto, Canada, ACM Press (2016)
6. <http://line-solver.sourceforge.net/>
7. <https://github.com/layeredqueueing/V5/tree/master/lqns>
8. Spinner, S., Casale, G., Zhu, X., Kounay S.: Librede: a library for resource demand estimation. In: ICPE, 227-228 ,2014.
9. A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In Proc. of the 3rd ICPE, 2012.
10. Cortellessa, V., Di Marco, A., Eramo, R., Pieantonio, A., Trubiani, C.: Approaching the Model-Driven Generation of Feedback to Remove Software Performance Flaws. In: EUROMICRO-SEAA, 162–169. IEEE Computer Society (2009)
11. Cortellessa, V., Di Marco, A., Trubiani, C.: Performance Antipatterns as Logical Predicates. In: Calinescu, R., Paige, R.F., Kwiatkowska, M.Z. (eds.) ICECCS, 146-156. IEEE Computer Society (2010)

12. Parsons, T., Murphy, J.: Detecting Performance Antipatterns in Component Based Enterprise Systems. *Journal of Object Technology* 7, 55–91 (2008)
 13. <http://cassandra.apache.org/>
 14. <http://hadoop.apache.org/>
 15. Smith, C.U., Williams, L.G.: More new software performance antipatterns: even more ways to shoot yourself in the foot. In: Computer Measurement Group Conference, 717–725 (2003)
-

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Iterative_Enhancement&oldid=3365184'

This page was last edited on 23 January 2018, at 11:39.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Fraud Detection

Contents

Context

Technologies

- Apache Spark
- Apache Cassandra

BigBlu : a Big Data platform to support fraud detection

- Overview
- Dashboard
- Detection module
- Administration
- Architecture
- Process: evolution of the platform
 - POC : proof of concept
 - MVP : minimum viable product
 - Version 1.0

DICE Tools

References

Context

This section will expose the purpose of fraud detection and its issues, then why and how Big Data could be a part of the solution, and finally an example of Big Data application helping fraud detection with BigBlu platform.

In the European Union, tax frauds represent over € 1,000,000,000,000(1 trillion) and “*it is more than the total spent on health across the EU*” according to Jose-Manuel Barroso, president of the European Commission between 2004 and 2014. Indeed countries have to spend significant sums on fraud detection and countermeasures and that reduces considerably the amounts granted for public services such as hospitals, schools and public transports. Tax frauds represent a huge problem for governments, causing them a big loss of money each year. For more than 145 governments impacted by this phenomenon, most of them suffering from repetitive economic crisis, the issue is the protection of billions of euros of revenue streams. However, when we step back to see the overall picture, we realize that tax fraud is not only about money. It is just the tip of the iceberg which hides many threats. Governments need to have a more efficient control on how money circulates, and, to a greater extent, how it is used.

Governments are increasingly using Big Data in multiple sectors to help their agencies manage their operations, and to improve the services they provide to citizens and businesses. In this case, Big Data has the potential to make tax agencies faster and more efficient. However, detecting fraud is actually a difficult task because of the high number of tax operations performed each year and the differences inherent in the way the taxes are calculated. The main measure of the EU against that is developing cooperation and exchange of information among governments in particular through IT tools. Thus, an application providing an instantaneous and secure access to all the tax information would speed up the fraud detection process. However some issues remain:

- Volume: with more and more taxpayers who generate each year more tax declarations, the data volume is strictly increasing;
- Variety: data come from multiple sources (electricity bill, local taxes information, etc.);
- Adaptability: new fraud types emerge regularly so the authorities have to adapt their detection method continuously;
- Security: it manipulates sensitive data, so it is not an option;

- Velocity, variability, volatility: the main aim is to do it quickly, and data arrive and change rapidly;
- Veracity, validity and value: because of the sensitiveness of data.

Big Data technology solves all the issues above. Indeed “*the appellation ‘Big Data’ is used to name the inability of traditional data systems to efficiently handle new datasets, which are too big (volume), too diverse (variety), arrive too fast (velocity), change too rapidly (variability and volatility), and/or contain too much noise (veracity, validity, and value).*” (ISO, Big data – Preliminary Report 2014).

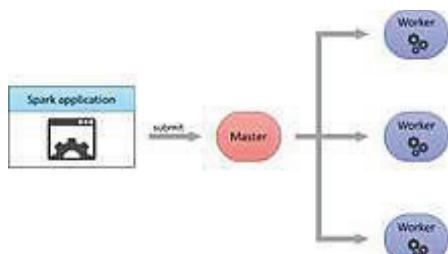
This application would not replace human detection but would bring out potential fraudsters to the authorities.

Technologies

Some technologies processing Big Data currently exist. This section will present some free open source Big Data technologies.

Apache Spark

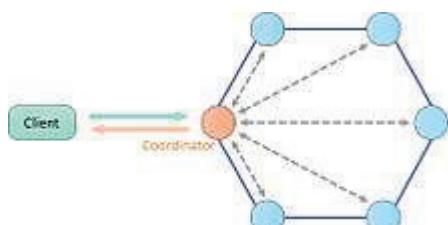
Apache Spark is an open source cluster-computing framework that performs complex analyses on a large scale, such as Big Data processing. Spark uses a very simple architecture with master and worker nodes, the whole constitutes a cluster that is qualified as vertical if the nodes are in separate machines or as horizontal if they all are in the same machine. Fundamentally a Spark application configures the job specifying the master information (port, IP address) and the job to execute. Then it submits it to the master node that connects to the cluster of workers to manage the resources and execute the job. Usually the job is implemented in an object-oriented programming language such as Java or Scala. The workers execute this job, so it has to be on each machine where there are workers. This constraint imposes a DevOps development because the developers need to know the infrastructure to set the configuration properly in the Spark application.



The main components in a Spark application

Apache Cassandra

Apache Cassandra is a free open source distributed NoSQL database management system. The nodes are organized in a ring and each one has an identical role and communicate with the others thanks to a peer-to-peer protocol. The data is stored as in key/value pair and is distributed into the different nodes thanks to a hash value of the key. Therefore, each node has a partition value to easily find data in the cluster



The ring structure of the nodes in Cassandra

When a client does a request, one node will become the coordinator and the client will communicate only with this node. The coordinator then works with the other nodes and sends the result to the client once the work is finished. Thus, while a node is the coordinator of a client, another node could be the coordinator of another client.

BigBlu : a Big Data platform to support fraud detection

BigBlu is a Big Data platform, developed as part of the DICE project, determining how to use Big Data technologies to support fraud detection. The main purpose was to explore tax declarations in order to extract potential fraudsters. Then, BigBlu also provides a scalable module to create and launch customized queries upon the database, to enable the authorities to adapt themselves to new types of fraud without further development.

Overview

BigBlu is divided into three main modules: the dashboard, the detection module and the administration module. This section will present each module

Dashboard

After authentication through connection interface, the user accesses the dashboard containing the table of jobs launched with their statutes and information about it such as the detection criteria used or the launch date.



The dashboard of BigBlu platform

The finished detections have a **Show Results** button (in the **Actions** column) that displays a table with all the people who meet the detection criteria and statistics. For example, there is a pie chart about gender distribution. Furthermore, some statistics panels give quick information about the number the average age and income of the potential fraudsters.

Detection module

That is important to distinguish detections and jobs. A detection is a user-created query with several criteria, a name and a description. A job corresponds to a detection launched on a database at a given time. Thus, all jobs are launched from the detection table: the user click on the **Launch** button of the detection he/she wants to launch, this displays a pop-up to select the database on which the detection will be launched and then redirects to the dashboard.



The detection table of platform
BigBlu

From the detection table, it is possible to edit, delete and create a detection. Only detections created by the logged in user are available. For example, the user *lparant* will not be able to access the detections of the user *yridene*, and vice versa. In addition, the detection criteria can be consulted simply by clicking on the detection line and will appear below the table in list form. The last version has four criteria available:

- Income decrease (percentage): select taxpayers whose income decreases more than the percentage chosen by the user compared to previous year;
- Income interval (two values);
- Gender (Male or Female);
- Age interval (two values).



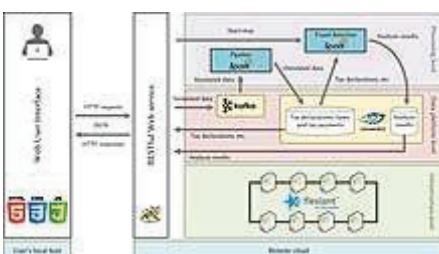
Criteria pop-up of BigBlu platform

The user can combine these four criteria to create custom queries. For example, he/she can make a query selecting women under 30 with an income between 5k and 10k.

Administration

Finally, the website administrator has access to pages to manage databases and users and information about the servers BigBlu uses to handle Big Data queries. Because administration pages do not use Big Data technologies, this section will be concise. In brief, there are two CRUD services: one for the databases and another for the users. Information about servers is static because the servers are determined by the application development and represent the Spark and Cassandra cluster

Architecture



Architecture diagram of BigBlu

Briefly, BigBlu is compound of three main parts:

- The user interface or Web client: a web application developed with HTML, CSS, jQuery and Bootstrap.
- The Big Data application: a Spark application developed in Java, using Spark API and Cassandra database.
- The Web service: the link between the Web client and the Big Data application. It also operates the management of users, databases and launched detections.

Process: evolution of the platform

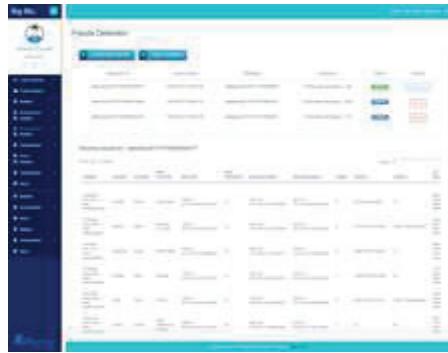
In order to explore the fraud detection through Big Data application we have used methods pointing out the feasibility and incremental featuring. Thus, we started with a POC (Proof of Concept), then a MVP (Minimum Viable Product) and finally the Version 1.0. We highly recommend this process because it allows the developers to discover safely and the technologies through the POC and then elaborate the product maximizing the value of what they have learnt during the last step to finally incrementally test, correct and complete the final product.

POC : proof of concept

First, a POC consists in implementing a demonstrator to prove that an idea is feasible. Therefore, we started by exploring the Big Data technologies such as Apache Spark and Apache Cassandra. Furthermore, we proceeded a technology watch to better understand Big Data and its issues. We recommend you the book **Principles and best practices of scalable real-time data systems** written by Nathan Marz and James Warren^[1], which explains the issues of processing Big Data and how to solve them.

MVP : minimum viable product

Then, we started to create an MVP with two types of indicators. The first one was about income decrease: when a taxpayer has a huge income decrease he/she is probably frauding by not declaring some incomes. The second one was when a taxpayer has low taxes with a high salary : there is potentially a fraud. During this step, we implement all the architecture with a basic interface. This tool could already perform Big Data queries and save their results in Cassandra.



The interface of BigBlu at MVP step

Version 1.0

The last version developed was an incremental enhancement of the MVP. First of all, the HMI was re-designed and we added a module to complete the evolutivity requirement: the new module detection now allows the creation of custom queries with a combination of four criteria. A CRUD service is implemented to enable the user to create, edit and delete his/her queries.

DICE Tools

DICE Tools	Benefit for the use case
DICE IDE	Useful hub for the other tools. Development activities and tools validation were made inside this IDE.
DICER	Rapid design of an execution environment by using concrete concepts. The various concepts specific to Cassandra and Spark are easier to understand with this graphical modelling language.
Deployment Service	Increased productivity This tool bears for us the burden to learn how to install and configure Cassandra and Spark on the cloud infrastructure.
Configuration Optimisation	By looking at the logs of Spark, this tool was able to find a better configuration.
Optimisation Tool	This tool is useful to evaluate the cost impact of the implementation of privacy mechanisms.
Simulation Tool	Given a set of requirements, a tax agency can use this tool to evaluate alternative architectures and measure the impact of business logic changes. The simulated models can also be used for documentation purpose.

References

1. NIST Big Data Public Working Group (2015-09). "NIST Big Data Interoperability Framework: Volume 1, Definitions. Final Version 1". NIST. doi:10.6028/NIST.SP.1500-1. https://bigdatawg.nist.gov/_uploadfiles/NISTSP.1500-1.pdf.

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Fraud_Detection&oldid=3367984'

This page was last edited on 28 January 2018, at 15:55.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Maritime Operations

Contents

Use Case Description

Business goals
Use Case Architecture

Use Case Scenarios

Deployment Scenario
Support vessels traffic increase for a given port
Add new business rules (CEP rules) for different ports
Give support to another port in the cloud instance of Posidonia Operations
Run a simulation to validate performance and quality metrics among versions

DICE Tools

Conclusion

Use Case Description

Posidonia Operations is an Integrated Port Operation Management System highly customizable that allows a port to optimize its maritime operational activities related to the flow of vessels in the port service area, integrating all the relevant stakeholders and computer systems.

In technical terms, Posidonia Operations is a real-time and data intensive platform able to connect to AIS (Automatic Identification System), VTS (Vessel Traffic System) or radar and automatically detect vessel operational events like port arrival, berthing, unberthing, bunkering operations, tugging, etc.

Posidonia Operations is a commercial software solution that is currently tracking maritime traffic in Spain, Italy, Portugal, Morocco and Tunisia, thus providing service to different port authorities and terminals.

The goals of creating this case study are adopting a more structured development policy (DevOps), reducing development/deployment costs and improve the quality of our software development process.

In the use case, the following scenarios are considered: deployment of Posidonia Operations on the cloud considering different parameters, support of different vessel traffic intensities, add new business rules (high CPU demand), run simulation scenario to evaluate performance and quality metrics.

Business goals

Three main business goals have been identified for the Posidonia Operations use case.

- Lower deployment and operational costs.

Posidonia Operations is offered in two deployment and operational modes: on-premises and on a virtual private cloud. When on-premises, having a methodology and tools to ease the deployment process will result in a shortened time to production, thus saving costs and resources. In the case of a virtual private cloud deployment, it is expected that the monitoring, analysis and iterative enhancement of our current solution will result in better hardware requirements specifications, which in the end are translated into lower operational costs.

- Lower development costs

Posidonia Operations is defined as a “glocal” solution for maritime operations. By “glocal” we mean that it offers a global solution for maritime traffic processing and analysis that can be configured, customized and integrated according to local requirements. In addition, the solution operates in real-time making tasks like testing, integration, releasing, etc. more critical. By the application of the methodology explained in the book, these tasks are expected to be improved in the development process, thus resulting in shortened development lifecycles and lower development costs.

- Improve the quality of service

Several quality and performance metrics have been considered of interest for the Posidonia Operations use case. Monitoring, predictive analysis or ensure reliability between successive versions will end in an iterative enhancement of the quality of service to our current customers.

Use Case Architecture

Posidonia Operations is an integrated port operations management system. Its mission consists on “globally” monitor vessels’ positions in real time to improve and automatize port authorities operations. The below image shows the general architecture of Posidonia Operations. The architecture is based on independent Java processes that communicate with each other by means of a middleware layer that gives support to a Message Queue, a Publication and Subscription API and a set of Topics to exchange data among components.

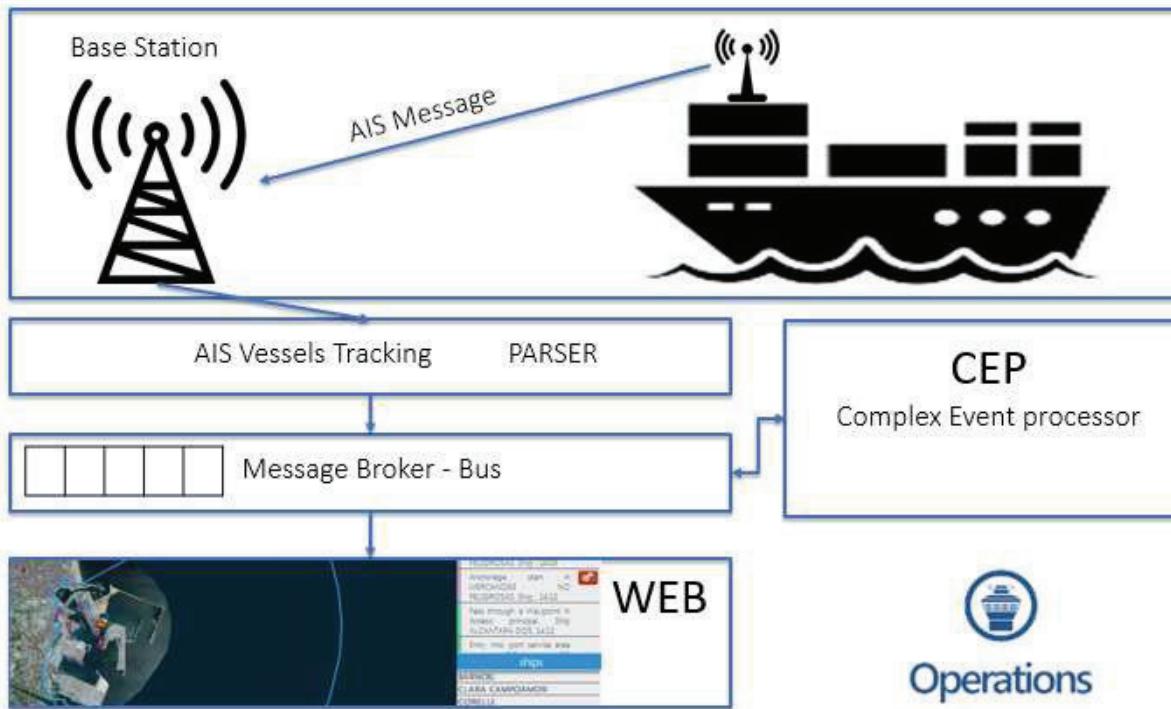


Figure: Posidonia Operations general architecture

An overview of the main components for Posidonia Operations would be:

- Vessels in the service area of a port send AIS messages that include their location and other metadata to a central station. (This is out of the scope of the architecture diagram)
- An AIS Receiver (a spout) receives those messages and emits them through a streaming channel (usually a TCP connection)
- The AIS Parser (a bolt) is connected to the streaming channel, parses the AIS messages into a middleware topic and publishes it to a Message Queue.
- Other components (bolts) subscribe to the Message Queue to receive messages for further processing. As an example, the Complex Event Processing engine receives AIS messages in order to detect patterns and emit events to a different Message Queue.
- The Posidonia Operation client "Web" allows to the employees of the port to have a visual tool that allows them to control the location of the vessels in real time. This website shows on a map the different vessels that are within the area of influence of a port, with a list of operations that are happening.

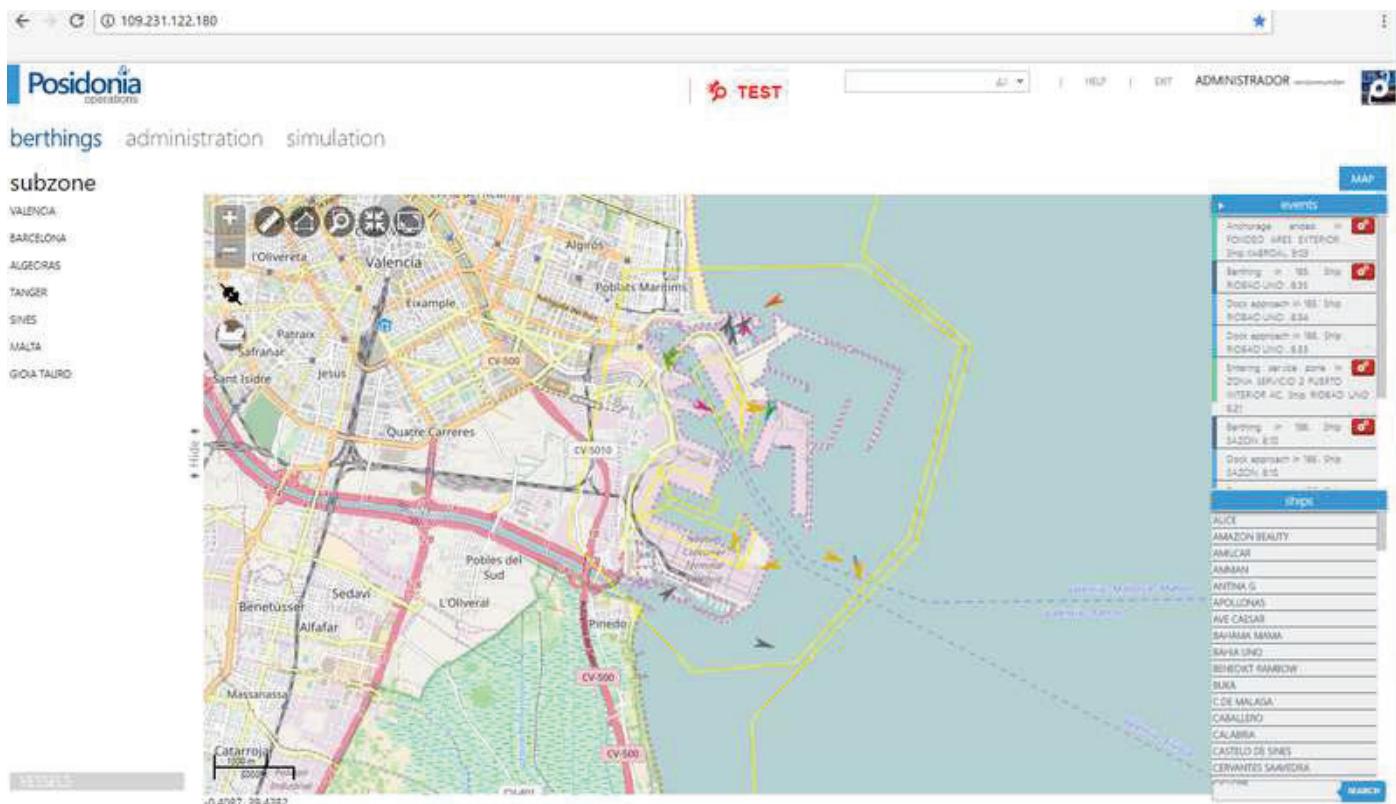


Figure: Posidonia Operations web client

Use Case Scenarios

There exists different usual scenarios where Posidonia Operations development lifecycle can benefit from the knowledge explained in this book. These scenarios are a small subset of the possible ones but are representative of interesting situations and are based on our current experience delivering a data intensive application to port authorities and terminals.

Deployment Scenario

Currently Posidonia Operations can be deployed in two fashions:

- On-premises: The port authority provides its own infrastructure and the platform is deployed on Linux virtual machines
- In the cloud: Posidonia Operations is also offered as a SaaS for port terminals. In this case, we use the Amazon Virtual Private Cloud (VPC) to deploy an instance of Posidonia Operations that gives support to different port terminals.

Apart from this, configuration varies depending on the deployment environment:

- Hardware requirements (number of nodes, CPU, RAM, DISK) to deploy of Posidonia Operations on each port is based on the team experience. For each deployment, the hardware requirements are calculated manually by engineers, considering the estimation of the number of vessels and the complexity of the rules applies for each message to be analysed. DICE tools can help to tune automatically the appropriate hardware requirements for each deployment.
- Posidonia Operations deployment and configuration is done by a system administrator and a developer and it varies depending on the port authority. Although deployment and configuration is documented, the DICE tools can help to adopt a DevOps approach, where deployment and configuration can be modelled in order not only to better understand the system by different stakeholders, but also to automate some tasks.
- A DevOps approach can help to provide also test and simulation environments that will improve our development lifecycle.

Support vessels traffic increase for a given port

Posidonia Operations core functionality is based on analysing a real-time stream of messages that represent vessels positions to detect and emit events that occur on the real world (a berth, an anchorage, a bunkering, etc.).

Different factors can make the marine traffic of a port increase (or decrease), namely:

- Weather conditions
- Time of the day
- Season of the year
- Current port occupancy

- etc.

This means that the number of messages per second to be analysed is variable and can affect performance and reliability of the events detected if the system is not able to process the streaming data as it arrives. When this is not possible, messages are queued and this situation has to be avoided. We currently have tools to increase the speed of the streaming data to validate the behaviour of the system in a test environment. However, the process of validating and tuning the system for a traffic increase is a tedious and time consuming process where DICE tools can help to improve our current solution.

Add new business rules (CEP rules) for different ports

Analysis of the streaming data is done by a Complex Event Processing engine. This engine can be considered as a “pattern matcher”. For each vessel position that arrives, it computes different conditions, that when satisfied produce an event.

The number of rules (computation) to be applied to each message can affect the overall performance of the system. Actually, the number and implementation of rules vary from one deployment to other

DICE tools can help on different quality and performance metrics, simulation and predictive analysis, optimization, etc. in order to tune our current solution.

Give support to another port in the cloud instance of Posidonia Operations

Give support to another port (or terminal) in the cloud instance of Posidonia Operations usually means:

- Increase the streaming speed (more messages per second)
- Increase on computation (more CEP rules executed per second)
- Deployment and configuration of new artefacts and/or nodes

In this case, DICE tools can help improve Posidonia Operations also on estimating the monetary cost of introducing a new port on the cloud instance.

Run a simulation to validate performance and quality metrics among versions

CEP rules (business rules) evolve from one version of Posidonia Operations to another. That means that performance and quality of the overall solution could be affected by this situation among different versions. Some examples of validations we currently do (manually):

- Performance: New version of CEP rules don't introduce a performance penalty on the system
- Performance: New version of CEP rules don't produce queues
- Reliability: New version of CEP rules provide the same output as prior version (they both detect the same events)

One of the main issues of the current situation is that measuring the performance (system performance and quality of the data provided by the application) is done manually and it's very costly to obtain an objective quantification. By using DICE simulation tools, performance and reliability metrics can be predicted for different environment configurations, thus ensuring high quality versions and non-regression.

DICE Tools

The DICE Framework is composed of several tools (DICE Tools): the DICE IDE, the DICE/UML profile, the Deployment Design (DICER) and Deployment service provide the minimal toolkit to create and release a DICE application. In order to validate the quality of the application the framework encompasses tools covering a broad range of activities such as simulation, optimization, verification, monitoring, anomaly detection, trace checking, iterative enhancement, quality testing, configuration optimization, fault injection, and repository management. Some of the tools are design-focused, others are runtime-oriented. Finally some have both design and runtime aspects and are used in several stages of the lifecycle development process.

Some of the DICE Tools have been used during the use case, in order to achieve the business goals set for the use case. The follow table summarizes the tools used during the use case and the benefits obtained from its use.

DICE Tools	Benefit for our Use Case
DICE IDE	The DICE IDE integrates all the tools of the proposed platform and gives support to the DICE methodology. The IDE is an integrated development environment tool, based on Eclipse, for MDE where a designer can create models to describe data-intensive applications and their underpinning technology stack. The DICE IDE integrates the execution of the different tools, in order to minimize learning curves and simplify adoption.
DICER	The DICER tool allows us to generate the equivalent TOSCA Blueprint (deployment recipe) from the Posidonia use case DDSM created using the DICE IDE. This Blueprint is used by the deployment service to automatically deploy the Posidonia use case. With this tool you can obtain different blueprints for different configurations of the use case.
Deployment Service	<p>Deploy the Posidonia Operations manually is quite time and cost consuming. Deployment Service is able to deploy on the cloud a configuration of the Posidonia Operations use case in few minutes. The last version of the deployment service works over flexiant Cloud orchestrator and Amazon AWS. To deploy a solution using the deployment service it is needed to provide as an input the TOSCA Blueprint of it. This blueprint is obtained using the DICER tool.</p> <p>Using Deployment service, the deployment is faster (only few minutes), you can deploy different configurations of your solution, and it does not require system administrator experts because the deployment is automatized.</p>
Monitoring Platform	<p>This tool allows us to obtain metrics in real time for a running instance of the Posidonia Operations: Generic hardware performance metrics (CPU and memory consumption, disk usage, etc.) and Specific use case metrics, such as, number of events detected, location of the events, execution time per rule, messages per second, etc.</p> <p>The reported results, provided by the monitoring platform, allow the architect and developers to eventually update the DDSM and/or DPIM models to achieve a better performance. Another interesting point is that the Monitoring Tools facilitates the integration with the Anomaly Detection Tool and the Trace Checking Tool, because both tools use the information stored in the monitoring as a data input.</p>
Fault Injection	<p>DICE Fault Injection Tool (FIT) is used to generate faults within virtual machines. In Posidonia use case, this tool is useful to check how the CEP component behaves against a high CPU load. To observe the behaviour of the system, we use the Monitoring Tool that contains a specific visualization for the system load.</p> <p>Although the Fault Injection Tools can launch other types of faults, for Posidonia use case, only the High CPU fault is relevant to evaluate the response of the system to this situation. It's important validate that the System continues working and no event is lost when a high load happens.</p>
Anomaly detection	Anomaly Detection Tool allows us to validate that the system works as expected with the current rules and with the addition of new ones. That is, no events are lost, no false positives are given, the execution time is kept within a reasonable range or the order of events detected is adequate.
Simulation Tool	<p>The vessels traffic increase for a given port directly affects the Posidonia Operations performance if the system is not properly sized. We have worked with the Simulation Tool and the Monitoring tool to define the dimension of the system and to monitor it in real time.</p> <p>Analysis of the streaming data is done by a Complex Event Processing engine. This engine can be considered as a "pattern matcher", for each vessel position that arrives it computes different conditions, that when satisfied produce an event. The number of rules (computation) to be applied to each message can affect the overall performance of the system. Actually, the number and implementation of rules vary from one deployment to other. In recent months, a significant effort has been made to improve the quality and the validation of the use case</p>

Conclusion

We affirm that the use of the DICE methodology and the DICE framework, are very useful in the Maritime Operations use case. It provides a productivity gain when developing the use case. The results obtained from applying the DICE tools to the use case can be summarized in:

- **Assessment of the impact in performance after changes in software or conditions** We can predict at design time the impact of changes in the software (number of rules, number of CEPs) and/or conditions (input message rate "Simulation Tool", CPU overloads "Fault Injection Tool"). Moreover, bottleneck and anomalies can be detected using the Anomaly Detection Tool.
 - **Increase the Quality of the system** We can detect punctual performance problems with the use of the Anomaly Detection Tool and we can detect errors in the CEP component. Detection of loss rules and false rules detection and delays in the detection of the rules compared to the real time in which the event happened.
 - **Automatic extraction of relevant KPI** Easy computation of application execution metrics (generic hardware metrics such as CPU, memory consumption, disk access, etc) and easy computation of application-specific metrics which have to do with computational cost of rules, number of messages processed per second, location of events on a map, quantification of application performance in terms of percentage of port events correctly detected by the CEP(s), etc.
 - **Automation of deployment** Much faster deployment and possibility of deployment in different cloud providers.
-

This page was last edited on 28 January 2018, at 10:56.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/News and Media

Contents

Use Case Description

Use Case Scenarios

“Trend Topic Detector” Detailed Description

Architecture

How to use the clustering module:

DICE Tools

Use Case Description

The news and media domain is a highly demanding sector in terms of handling large data streams coming from the social media. ATC SA, as one of the leading brands around the world in ICT application for the news and media domain, has developed NewsAsset – a commercial product positioned in the news and media domain. NewsAsset is a commercial product positioned in the news and media domain, branded by Athens Technology Center, an SME located in Greece. The NewsAsset suite constitutes an innovative management solution for handling large volumes of information offering a complete and secure electronic environment for storage, management and delivery of sensitive information in the news production environment. The platform proposes a distributed multi-tier architecture engine for managing data storage composed by media items such as text, images, reports, articles, videos, etc. Innovative software engineering practices, like Big Data technologies, Model-Driven Engineering (MDE) techniques, Cloud Computing processes and Service-Oriented methods have penetrated in the media domain. News agencies are already feeling the impact of the capabilities that these technologies offer (e.g. processing power, transparent distribution of information, sophisticated analytics, quick responses, etc.) facilitating the development of the next generation of products, applications and services. Especially considering interesting media and burst events which is out there in the digital world, these technologies can offer efficient processing and can provide an added value to journalists. At the same time, heterogeneous sources like social networks, sensor networks and several other initiatives connected to the Internet are continuously feeding the world of Internet with a variety of real data at a tremendous pace: media items describing burst events, traffic speed on roads; slipperiness index for roads receiving rain or snowfall; air pollution levels by location; etc. As more of those sources are entering the digital world, journalists will be able to access data from more and more of them, aiding not only in disaster coverage, but being used in all manner of news stories. As that trend plays out, when a disaster is happening somewhere in the world, it is the social networks like Twitter, Facebook, Instagram, etc. that people are using to watch the news ecosystem and try to learn what damage is where, and what conditions exist in real-time. Many eyewitnesses will snap a disaster photo and post it, explaining what's going on. Subsequently, news agencies have realized that social-media content are becoming increasingly useful for disaster news coverage and can benefit from this future trend only if they adopt the aforementioned innovative technologies. Thus, the challenge for NewsAsset is to catch up with this evolution and provide services that can handle the developing new situation in the media industry. In addition, during DICE project we have identified a great business opportunity on the “Fake News” sector. More specific, we have used DICE tools in order to develop specific modules (part of the News Orchestrator application) for a new and innovative product which is being connected via an API to our NewsAsset suite or can be sold as a standalone solution. This new product is being called TruthNest (www.truthnest.com). TruthNest is a service ATC has implemented for assessing the trustworthiness of information found in Social Media. TruthNest users are able to capture streams from social networks, from which they are then able to analyse a single post and gain insights according to several dimensions of a verification process.

Use Case Scenarios

TruthNest is an online comprehensive tool that can promptly and accurately discover, analyse, and verify the credibility and truthfulness of reported events, news and multimedia content that emerge in social media in near real time. The end user has the ability to verify the credibility of a single post within seconds by activating, with a single click, a series of analysis events for achieving the desired result. More specific, TruthNest users will be able to bring in streams from social networks which will then be able to analyse and gain insights as to several dimensions of the verification process. In addition, they will also be able to create and monitor new “smart” streams from within TruthNest.

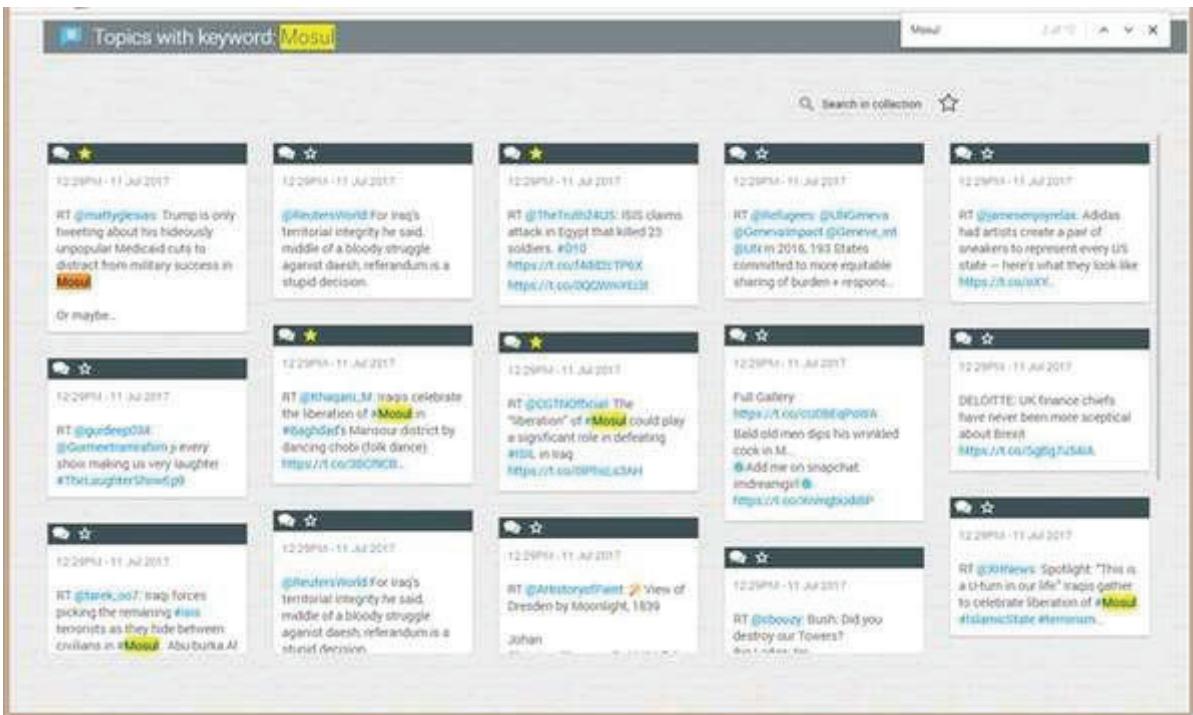


TruthNest Screenshot

An important module for TruthNest, which has been developed from scratch, is the “Trend Topic Detector”. The “Trend Topic Detector” provides to the end user a visualisation environment showing prominent trends that derive from social media, and, more specifically, from Twitter. What is critical to mention at this stage, is that only the “Trend Topic Detector” module has been developed by using DICE tools while the rest of TruthNest’s components have been developed by using conventional tools and methodologies as these have been used by ATC’s engineering and development team.

“Trend Topic Detector” Detailed Description

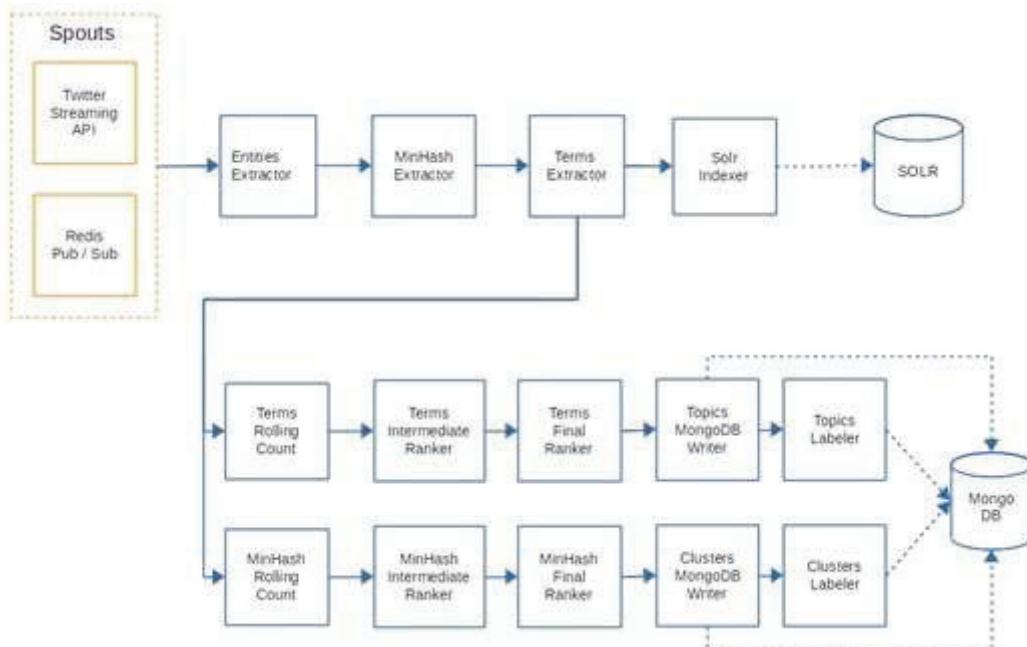
The Trend Topic Detector is centered around a clustering module. This creates clusters of tweets that relate to the search criteria submitted. The clusters are formulated by grouping the tweets found based on their common terms. The module tracks a percentage of the tweets posted onwards on Twitter, as the Twitter streaming API limitations impose. While it is restricted currently on Twitter stream API, it can take input from multiple social media (YouTube, Flickr and others) however it has not been implemented yet.



Topic Detector Screenshot

Architecture

The main pipeline of the clustering module is implemented as a Storm topology, where sequential bolts perform specific operations on the crawling procedure. These bolts include entity extraction (by using Stanford NER classifier) and minHash computation to estimate how similar the formulated sets are. The tweets terms are extracted and indexed in a running Solr instance. The most frequent terms are computed in a rolling window of 5 minutes and 20 clusters are formulated by default. A label (typically the text of a tweet) is assigned to each cluster. The results are stored in a Mongo database. The module is highly configurable and offers nearly real time computation of clusters.



Topic Detector Topology

How to use the clustering module:

1. The user sets search terms through the user interface. The default language is English, other languages are not officially supported. Some trending topic cluster computation settings are also available (e.g. window computation

time).

2. The process is initialized and the user is informed on the number of tweets currently analysed. A diagram (line chart) is shown that is renewed every few seconds, showing the stream progress. After 10 minutes if tweets were found, the first trending topic clusters are presented. A set of 20 trending topic clusters is shown. The trending topic cluster are clickable and the user can view the items that consist them.
3. The trending topic clusters are re-computed every five minutes and their content is updated. The user can view details (e.g. clusters evolution through time).
4. The user can search the trending topic clusters and tag the most important ones. A favourite filter is also available.
5. The user can start/stop a trending topic cluster and delete it. He can save a trending topic cluster and restart it at a later time.
6. There are limits on the number of trending topic clusters created and their activity period. Typically, the clusters are stopped after 24h.

DICE Tools

DICE Tools	Benefit for our Use Case
DICE IDE	The fact that most DICE tools is planned to be progressively integrated in a common place, the DICE IDE, makes the interaction between the various DICE tools really effective since most tools depend on the DICE Profile models. For example, the DICE Verification tool expects a DTSM model, properly annotated with some quality characteristics regarding the level of parallelization of the NewsAsset DIA, as input and the verification process can be triggered from within the DICE IDE. In this way the DICE IDE not only covers the design time modelling requirements for our DIA but it effectively links the outcome artefacts to the tools related to other phases of a DevOps approach, for example deployment and monitoring.
DICER	The DICER tool allowed us to express the infrastructure needs and constraints for the NewsAsset application and also to automatically generate deployment blueprints to be used on a cloud environment. We evaluated the usefulness of the DICER tool in terms of time saving, with regard to the time needed to setup the infrastructure manually from scratch, and the degree of automation that DICER offers. Note that in the total time that we computed for the DICER execution time we included the time needed by the DICE Deployment service to deploy the generated blueprint.
Deployment Service	The whole process was really fast and we achieved a time saving almost 80% compared to the time we needed previously when we were installing manually the Storm cluster and all of its dependencies (Zookeeper etc) as well as the Storm application and all the dependencies for the persistence layer. The fact that the TOSCA blueprints allow the refinement of the Storm-specific configuration parameters in advance is really convenient since we can experiment with different Storm cluster setups by applying another reconfiguration, resulting in a new testbed, until we reach the most efficient in terms of performance and throughput for our topologies.
Monitoring Platform	The ability to monitor NewsAssets deployed topologies is necessary in order to identify any possible bottleneck or even to optimize the performance and throughput by adding more parallelization for example. We have installed the Monitoring platform core modules (ELK stack, DMon core), and we then use one of the features of latest Deployment service release to automatically register the Storm cluster nodes on the core services of Monitoring platform during the infrastructure/application deployment phase. The monitoring can be performed not only on a per (Stormcluster) node but also on a per application level which means that we can distinguish between issues related to hardware specifications of the nodes and issues related to the application's internal mechanics like the size of internal message buffer of Storm.
Fault Injection	Since the NewsAssets topologies deal with a cloud deployment it is quite critical to be able to test the consequences of faults early in the development phase in a controlled environment rather than once the application is in production. It is important for the NewsAsset DIA to be comprised of reliable topologies due to the nature of the processing it performs: if for example there is a burst event at some time then losing some of the social networks messages due to network failures/repartitions could affect the quality of the trending topics identified at that period. Thus, having a tool like Fault Injection can help us to test how the application behaves in terms of reliability and fault-tolerance. There is also the need for the NewsAsset application to eliminate any single point of failure as possible. We used the Fault Injection tool to randomly stop/kill not only the various types of Storm processes (nimbus, supervisor and worker processes) but also a whole node of the Storm cluster and we checked how those actions affect the proper execution of the topologies. Finally, we used the Fault Injection tool to generate high memory and CPU usage for the Storm cluster VMs to simulate high memory/CPU load. The various processing bolts of the topology were not affected too much since none of them is high CPU-bound or memory greedy. We plan to continue experimenting with the Fault Injection tool specially by simulating high bandwidth load since the trending topic detector topology makes heavy use of the Twitter Stream API.
Quality Testing	The validation KPI for the Quality Testing Tool refers to more than 30% reduction in the manual time required in a test cycle. For that purpose, a chrono assessment has been performed comparing the manual time that ATC engineers would need to perform manually the stress testing of the topology to the time that is required by the Quality Testing tool to perform a similar task. In our use case scenario only 5 iterations/experiments the goal has been achieved and the reduction per test cycle has reached the 34%. In case that more test cycles are required the reduction can get even higher. Another important finding is that the load testing process can be fully automated by using the Quality Testing Tool.
Configuration Optimisation	In general, a typical Storm deployment comprises of a variety of configuration properties that affect the behaviour of nimbus, supervisors as well as the topologies. It is a non-trivial task for a developer to select optimal values for the configuration properties in order to fine tune the Storm topology execution. This is usually a complicated task accomplished by experts in this area. Also, each Storm based application has its own characteristics (either I/O bound or CPU bound or even both) that should be taken carefully into account while experimenting with candidate optimal values, so following a generic template for setup is not an option. Also, the overhead

and the time required to update the configuration and reload it on the cluster each time one tries to optimize some properties is significant.

In order to evaluate the improvement by applying the Configuration Optimization tool on the Storm configuration that the News Orchestrator relies on, the ATC engineers have monitored the topology throughput as well as the latency. The impact on performance is more than twice compared to the default configuration which is a significant improvement. This achievement has been achieved after only 100 iterations which resulted in a total of 16 hours (100 * 10 minutes) of execution. The corresponding validation KPI has been fully addressed, resulting in a more than 30% improvement in both the throughput and the latency metrics.

Scalability, bottleneck detection and simulation/predictive analysis are some of the core requirements for the News Orchestrator DIA. The DICE Simulation tool promises that it can perform a performance assessment of a Storm based DIA that would allow the prediction of the behaviour of the system prior to the deployment on a production cloud environment. The News Orchestrator engineers are often spending much time and effort in order to configure and adapt the topology configuration according to the target runtime execution context. Introducing a tool that can perform such a demanding task efficiently would clearly increase the developer's productivity and also facilitate their testing needs. The corresponding validation KPI of the Simulation tool refers to a prediction error of the utilization of bolts less than 30%. The comparison should be performed between the results that the Simulation tool has generated and those monitored by the metrics interface that Storm framework exposes. In this way the predicted/simulated results were validated against the actual results that were monitored by deploying the News Orchestrator topology on a Storm cluster of 4 nodes running on the cloud.

The results show that for some of the bolts (approximately more than half) the prediction error is indeed very small, less than 10%, predicting quite accurately the capacity of the bolts. What is interesting is what are the characteristics of the bolts that the Simulation tool fails to predict their capacity, or at least with an acceptable precision. There is a tricky part in the workflow of the bolts of the News Orchestrator topology: some bolts are not constantly consuming and analysing social items but instead their execution is triggered periodically whenever a predefined time window expires (that time window reflects how frequently the News Orchestrator engineers want to detect news topics, which is configured at 5 minutes). So, it was proved that for the topic detection and minhash clustering bolts the average execution time was hard to be computed, affecting thus the precision of the Simulation tool results. Additionally, some of those bolts are by nature of low utilization mostly due to the limited time they are executed compared to the total experiment time. In those cases, is also observed a deviation between the Simulation tool results and the actual monitored values that surpasses the threshold of 30% regarding the predictions. For the ATC engineers, having a tool like the DICE Simulation tool in the stack of the tools that help them to optimize and evaluate the News Orchestrator DIA is a significant advantage. Every new feature that is added in the News Orchestrator DIA may result in an undesired imbalance with regards to the performance of the system. Being able to validate the performance impact on the DIA prior to the actual deployment on a cloud infrastructure gives the flexibility to fine tune the topology configuration in advance and take corrective actions (i.e. scaling by increasing bolts parallelism) without wasting resources (costs and efforts) that would be otherwise required by an actual deployment on the cloud.

Simulation
Tool

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/News_and_Media&oldid=3339518

This page was last edited on 6 December 2017, at 15:07.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Future Challenges

Although the book has proposed a concrete solution for practical DevOps in software engineering, unavoidably the book could not cover in details a number of emerging themes in DevOps and software systems development. In this section, we discuss two emerging challenges in the area, which may spawn new industrial and academic research efforts.

From DevOps to DataOps

Throughout the book, we have focused on developing enterprise IT applications that rely on Big data processing technologies. However, we have not developed in detail the problem of designing and maintaining a data feeding pipeline for a Big data application. This problem, which typically falls under the remit of data engineering experts, involves the acquisition, filtering, analysis, and storage of the data collected by the Big data application. Specific architectures are progressively emerging, such as the notion of *data lakes* that consolidate all heterogeneous datasets of an organization in a single storage location, and data engineers have recently recognized the need to adopt DevOps-style methods for the release and update of the code involved in the data pipelines.

To make a concrete example, a set of teams collaborating on developing a business analytics pipeline, also need to test, update, and release new data queries and statistical analyses, in the same way in which a software engineer releases new versions of an application or its composing elements. This emerging trend, called *DataOps*, raises new research challenges, such as deciding which methods developed in DevOps could be adopted in this problem domain or establishing whether DevOps and DataOps methods can coexist in the same methodology. Quality-driven engineering methods raise strong challenges in the DataOps domain, as one needs to guarantee data privacy and data integrity with all releases, calling for dedicated methods for testing.

DevOps and Micro-services

Emerging industry trends, such as microservices, are raising the problem extending DevOps to support the delivery of cloud applications that are decomposed into fine-grained and independently deployable services, called micro-services. Micro-services different from traditional web services in a number of ways:

1. Micro-services are *lean*, meaning that a traditional web service can expose many APIs, whereas a micro-service typically implements a single, or very few functions. Moreover, a micro-service is typically based on lightweight REST/JSON communication, as opposed to the SOAP/WS-* stack common in web services.
2. Next, a micro-service is meant to map exclusively to a single development team, so that each team can release *independently* of the others new versions of that service. This implies, among others, the notion that each team can manage in a decentralized fashion its own data, relieving the application architecture of shared databases and thus fostering the need to define new architectural paradigms and also new methods to ensure availability, consistency, and checkpointing. Another important property of micro-services is
3. Lastly, and perhaps most importantly, micro-services are meant to exploit the advantages of containerization, which implies, for example, the ability to autoscale the services. A consequence of this is that when a micro-service becomes so fine-grained to be equivalent to a single function call in the application code, a so-called *mono-service*, it is effectively possible to auto-scale portions of the application code, trading performance for a lower cost of scaling.

Clearly, the above properties of micro-services make them rather different from basic enterprise applications, and they require DevOps to evolve in the direction of being a decentralized practice, with each team having at its disposal tools to reason about application updates before delivering to production. Developing support tools for decentralized software engineering of microservices and to reason on how to decompose an application architecture into right-sized services are among the most pressing research challenges for the near future.

This page was last edited on 26 January 2018, at 10:01.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Closing Remarks

This book has been motivated by the problems that organization face today in developing Big Data systems. Today, most organizations face high market pressure, and their supporting ICT departments are struggling to accelerate the delivery of applications and services while preserving production and operations stability. On the one hand, ICT operators lack understanding of the application internals including system architecture and the design decisions behind architectural components. On the other hand, development teams are not aware of operation details including the infrastructure and its limitations and benefits. These issues are even magnified for Big Data systems. For these, recent years have seen the rapid growth of interest for the use of data-intensive technologies, such as Hadoop/MapReduce, NoSQL, and stream processing. These technologies are important in many application domains, from predictive analytics to environmental monitoring, to e-government and smart cities. However, the time to market and the cost of ownership of such applications are considerably high.

The book has explained how DevOps practices can address the common isolation between Development and Operations. We have illustrated the DICE methodology, a practical approach to software engineering of Big Data system leveraging the idea that design artefacts, such as UML and TOSCA models, can play a pivotal role in organizing the architectural and requirements knowledge within an organization and share it across the design-time and run-time tools of the DevOps toolchain. Contrary to most DevOps tools available in commerce, which narrow the focus on continuous integration and continuous delivery, DICE thus implements a more ambitious form of DevOps, in which the unification of Dev and Ops is not limited only to the delivery phase, but the whole lifecycle of the application development, with both developer and operators relying on the models to carry out their activities. By having a DevOps toolchain that covers the entire application lifecycle, including feed-back of production monitoring data to designers and developers, the proposed DevOps methodology promotes iterative enhancement of an application driven by quantitative monitoring data.

One of the lessons we have learned in defining the DICE methodology is that different actors have very different expectations on the role that the models should play. Designers see models as a core element of their work, therefore they can find natural to work with model-driven engineering tools. The same does not come natural to most operators, which use abstractions closer to the operating system and the system administration way of thinking. Singularly, if the models are presented in textual form, as in the case of TOSCA orchestration and topology models, operators tend to be at ease with them. This apparent contradiction reveals that, even though a methodology such as DICE can be fully based on models behind the scenes, it is important that different levels of exposures of the models is given to different actors in the toolchain, in order for them to be at ease with the tools they use. Resolving such cultural differences between Dev and Ops indeed represents one of the main challenges of delivering a sound DevOps methodology. We believe that the DICE methodology present in this book represents one of the first instances of methodology to develop data-intensive applications that attempts to systematically tackle this problem.

Retrieved from 'https://en.wikibooks.org/w/index.php?title=Practical_DevOps_for_Big_Data/Closing_Remarks&oldid=3367146'

This page was last edited on 26 January 2018, at 10:03.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).

Practical DevOps for Big Data/Glossary

Acronym	Meaning
CEP	Complex Event Processing
DIA	Data-Intensive Applications
DoE	Design of Experiments
DPIM	DICE Platform Independent Model
DSL	Domain-Specific Language
DTSM	DICE Platform and Technology Specific Model
DDSM	DICE Deployment Specific Model
EMF	Eclipse Modeling Framework
GSPN	Generalized Stochastic Petri Net
GUI	Graphic User Interface
HC	Hill Climbing
HDFS	Hadoop distributed file system
IDE	Integrated Development Environment
MDD	Model Driven Development
MDE	Model-Driven Engineering
MMAP	Marked Markovian Arrival Process
MOF	Meta-Object Facility
M2M	Model-to-Model
M2T	Model-to-Text
MTBF	Mean Time Between Failures
MTTF	Mean Time to Failure
MTTR	Mean Time To Repair
NFP	Non-Functional Property
OASIS	Advanced open standards for the infomation society
OMG	Object Management Group
QoS	Quality of Service
QT	Quality Testing
QT-GEN	QT workload generator
QT-LIB	QT library
RDD	Resilient Distributed Dataset
REST	Representational state transfer
SLA	Service-Level Agreements
SWN	Stochastic Well-formed Net
TOSCA	Topology and Orchestration Specification forCloud Applications
UML	Unified Modeling Language
VSL	Value Specification Language
WS	Web Services
YAML	YAML Ain't Markup Language

This page was last edited on 5 January 2018, at 16:09.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).