# **Architecture Components for Big Data Processing**

Architecture components for big data processing depend on the specific technologies and tools used in a given system. However, the following are some examples of common architecture components for big data processing:

Data ingestion components:

* Apache Kafka: a distributed streaming platform used to collect and process large volumes of data in real-time.
* Apache Flume: a distributed data collection and aggregation system used to efficiently ingest and transfer large volumes of log data.

Data processing components:

* Apache Spark: a fast and general-purpose distributed computing system for large-scale data processing.
* Apache Hadoop MapReduce: a distributed computing framework used to process large amounts of data in parallel.

Data storage components:

* Apache Hadoop Distributed File System (HDFS): a distributed file system used for storing and processing large volumes of data.
* Apache Cassandra: a distributed NoSQL database used for storing and managing large volumes of structured and unstructured data.

Data access components:

* Apache Zeppelin: an open-source web-based notebook for data analysis and visualization.
* Apache Drill: a distributed SQL query engine that supports complex queries across multiple data sources.

Data governance components:

* Apache Atlas: a scalable and extensible metadata management system that enables organizations to manage their data governance requirements.
* Apache Ranger: a framework for managing security policies across the Hadoop ecosystem.

Infrastructure components:

* Cloud infrastructure: such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform, which provide scalable computing and storage resources on-demand.
* High-performance computing clusters: such as Apache Mesos or Kubernetes, which enable efficient use of computing resources for big data processing.

These concrete architecture components work together to provide a robust, scalable, and efficient big data processing system that can handle large volumes of data and provide valuable insights and information for organizations.

# **Big data streaming architecture**

Concrete big data streaming architecture involves processing and analyzing large volumes of data in real-time or near-real-time. The following are some key components of a big data streaming architecture:

* Data sources: Data sources can include sensors, log files, social media feeds, and other real-time data streams.
* Data ingestion: Data ingestion components such as Apache Kafka or Apache Flume can be used to collect and store the streaming data.
* Data processing: Data processing components such as Apache Spark Streaming or Apache Flink can be used to process the streaming data in real-time, enabling near real-time insights.
* Data storage: Data storage components such as Apache Cassandra or Apache HBase can be used to store the processed data.
* Data access: Data access components such as Apache Drill or Apache Superset can be used to provide users with access to the processed data.
* Data governance: Data governance components such as Apache Ranger or Apache Atlas can be used to manage the security and compliance requirements of the streaming data.
* Infrastructure: Infrastructure components such as high-performance computing clusters or cloud infrastructure can be used to provide scalable computing and storage resources for the streaming data.

# **System development**

System development is the process of designing, developing, testing, and deploying software systems to meet specific business requirements. The following are the key steps involved in the concrete system development process:

Requirements gathering: The first step in the system development process is to gather requirements from stakeholders. This involves conducting interviews and workshops to understand the business needs and document the system requirements.

System design: Once the requirements are gathered, the system design phase involves creating a detailed architecture and design for the system. This includes defining the system components, data models, and user interfaces. This phase may involve the use of tools such as Unified Modeling Language (UML) diagrams, data flow diagrams, and wireframes.

Development: The development phase involves writing code to implement the system design. This includes programming, testing, and debugging. The development phase may involve the use of programming languages such as Java, Python, or Ruby, and software development tools such as integrated development environments (IDEs), code editors, and version control systems.

Testing: Once the code is developed, it needs to be tested to ensure that it meets the business requirements and is free of bugs and errors. This involves unit testing, integration testing, and system testing. The testing phase may involve the use of automated testing tools such as Selenium, JUnit, or TestNG.

Deployment: Once the system is tested and validated, it can be deployed to the production environment. This involves configuring the system on the target infrastructure and ensuring that it operates reliably in a production environment. This phase may involve the use of deployment tools such as Ansible, Chef, or Puppet.

Maintenance: Once the system is deployed, it needs to be maintained over time to ensure its continued reliability and scalability. This includes updating software dependencies, optimizing code, and upgrading hardware as needed. The maintenance phase may involve the use of monitoring tools such as Nagios, Zabbix, or Prometheus.

## **1. Conception phase**

In this project, the goal is to design and implement a data architecture for a data-intensive large-scale application. The project requires adaptation of principles such as Infrastructure as Code (IaC) and Microservices (MS) to build the backend of data-intensive applications using state-of-the-art concepts and methods. The project aims to design a reliable, scalable, and maintainable batch-processing data system for a data-intensive machine learning application. The designed system must be able to ingest massive amounts of data, store the data effectively, pre-process parts of it, and aggregate the data for direct usage in a machine learning application. The data infrastructure will be designed in a way that data processing is scheduled to be conducted in batches, and data engineering principles will be adapted during the process.

Software Components.

To build a data-intensive application, familiarizing oneself with common software components is necessary. In this project, the following software components will be used:

Apache Kafka: Kafka is a distributed streaming platform used for building real-time data pipelines and streaming applications. Kafka can handle high-volume, high-throughput, and low-latency data streams, making it ideal for real-time data processing.

Hadoop: Hadoop is an open-source framework used for storing and processing large datasets across distributed computing clusters. Hadoop provides a reliable, scalable, and fault-tolerant storage layer for big data.

Spark: Spark is an open-source big data processing engine used for processing large datasets. Spark provides an easy-to-use API and high-performance processing capabilities.

The benefits of using these software components are that they are widely adopted and have a large developer community, providing continuous support, updates, and improvements. These software components are also open-source, which makes them cost-effective.

The trade-off is that these software components can be complex to set up and manage. They require a certain level of expertise to work with effectively. Moreover, since these components are open-source, security vulnerabilities can be a concern.

Reliability, Scalability, and Maintainability:

To ensure reliability, scalability, and maintainability of the designed system, the following techniques will be used:

* Microservices: The system will be designed using microservices architecture. Microservices will enable each service to work independently and in isolation, providing better fault isolation and easier maintenance.
* Containerization: Docker will be used to containerize the system components, making the system more scalable and portable. Containerization also enables faster deployment and rollbacks, making it easier to maintain the system.
* Infrastructure as Code (IaC): IaC will be used to automate the infrastructure deployment process, making it more reliable and less prone to human errors.

These techniques ensure that the system is reliable, scalable, and maintainable. The microservices architecture enables better fault isolation, making it easier to maintain the system. Containerization makes the system more scalable and portable, enabling faster deployment and rollbacks. IaC automates the infrastructure deployment process, reducing human errors, and making the process more reliable.

Data Security, Governance, and Protection:

To ensure data security, governance, and protection, the following techniques will be used:

* Authentication and Authorization: Authentication and authorization mechanisms will be used to control access to the system and its data. This will prevent unauthorized access to the system and data.
* Data Encryption: Data encryption will be used to protect the confidentiality and integrity of the data. Encryption will ensure that the data can only be accessed by authorized users.
* Data Governance: Data governance policies will be implemented to ensure that the data is used ethically and in compliance with legal and regulatory requirements.

These techniques ensure that the data is secure, governed, and protected. Authentication and authorization mechanisms will ensure that only authorized users can access the system and its data, while data encryption will protect the confidentiality and integrity of the data. Data governance policies will ensure that the data is used ethically and in compliance with legal and regulatory requirements. All of these techniques are crucial for maintaining the security, governance, and protection of the data and the system as a whole.

In addition to these techniques, other measures such as regular backups and disaster recovery plans will also be implemented to ensure the availability and recoverability of the data in case of any unforeseen events or disasters. These measures will help to minimize the risk of data loss or downtime, and ensure that the system remains operational and reliable at all times.

Questions:

Which microservices will take care of data ingestion to my system?

For data ingestion, we can use microservices that are responsible for capturing and processing data from various sources such as APIs, databases, files, and message queues. Some examples of microservices that can handle data ingestion are:

1. Data collection service: This service is responsible for collecting data from various sources and feeding it into the system.
2. Data processing service: This service processes the raw data into a usable format by cleaning, transforming, and aggregating it.
3. Data validation service: This service validates the incoming data to ensure that it meets the expected format and data quality standards.
4. Data storage service: This service stores the ingested data into a database or data lake for future use.

The exact combination of microservices will depend on the specific requirements and architecture of the system. It is essential to design a modular and scalable system to handle increasing data volumes and changing data sources over time.

Which microservices will take care of data pre-processing and aggregation?

To handle data pre-processing and aggregation, we will use microservices such as Apache Spark and Apache Flink. Apache Spark provides a distributed computing framework for data processing and allows for parallelization of data processing tasks. Spark also provides various libraries for pre-processing data, such as Spark SQL and Spark Streaming. Apache Flink is another distributed computing system that can handle stream and batch data processing. Flink is known for its ability to handle real-time streaming data processing efficiently.

Which microservices will take care of data storage in my system?

One microservice that could take care of data storage in your system is a database management system such as MongoDB or PostgreSQL. This microservice would be responsible for storing the data ingested by the data ingestion microservice in a way that allows for efficient querying and retrieval. Additionally, this microservice could be configured to implement data replication and backup strategies to ensure data redundancy and fault tolerance. Another option could be a distributed file system like Hadoop Distributed File System (HDFS) or Amazon S3, which allows for scalable storage of large volumes of data.

Which microservices will take care of data delivery to the frontend machine learning application?

The microservices responsible for delivering data to the frontend machine learning application will depend on the specifics of the application. However, we could use Apache Kafka for real-time data streaming, or an API for retrieving processed data. Apache Kafka is a distributed event streaming platform that can handle high-throughput data streams. It can also handle real-time data processing and real-time analytics. APIs, on the other hand, provide a more structured way of delivering data to the frontend, and can be used for data delivery in a more controlled manner.

Which techniques will I use to implement reliability, scalability, and maintainability to my system?

To implement reliability, scalability, and maintainability, we will use techniques such as containerization, automation, and monitoring. Containerization allows for a more modular and scalable approach to deploying microservices. We will use Docker to create containers for each microservice, making them portable and easier to manage. Automation will also be used to streamline the deployment and management of microservices. We will use Infrastructure as Code (IaC) to automate the provisioning and configuration of our infrastructure. Monitoring will also be implemented to ensure the health of our system. We will use tools like Prometheus and Grafana to monitor system performance and detect and resolve issues before they become critical.

Which techniques will I use to ensure data security, governance, and protection?

To ensure data security, governance, and protection, we will implement access controls, encryption, and regular backups. Access controls will be used to limit access to sensitive data to authorized personnel only. Encryption will be used to protect data in transit and at rest. Regular backups will also be implemented to ensure that data can be recovered in case of a disaster. We will use tools like HashiCorp Vault to manage secrets and provide secure access to sensitive information. We will also follow security best practices and standards such as the OWASP Top 10 and ISO/IEC 27001.

Which docker images will I use to build the system and must they be modified?

The docker images we will use to build the system will depend on the specific microservices we choose to use. For example, we could use the official Apache Kafka and Apache Spark images from Docker Hub. We may also use other images for specific components such as databases, caching systems, and load balancers. We will modify these images as necessary to include any custom configurations and dependencies needed for our application.

Which data will I use for my project?

For our project, we will choose a data source that contains several data points and is time-referenced, i.e., containing a timestamp for each data point. A good starting point for finding such a data source is Kaggle, where numerous open sample datasets are available. We will choose a dataset that is relevant to our use case, and contains data that can be used for training and testing a machine learning model.

At which frequency will my system ingest (e.g. monthly), process, aggregate and deliver data (e.g. quarterly)?

The system will ingest data on a daily basis, process and aggregate the data on a weekly basis, and deliver the pre-processed data to the frontend machine learning application on a monthly basis. This frequency of data processing and delivery strikes a balance between keeping the machine learning model up-to-date while not putting undue stress on the system resources. The exact frequency can be adjusted based on the specific needs and requirements of the application.

The tech stack for the designed data-intensive large-scale application could include the following components:

Backend:

* Microservices architecture
* Apache Kafka for real-time data streaming
* Hadoop for distributed storage and processing of large datasets
* Apache Spark for processing large datasets
* Python or Java as the programming language
* Docker for containerization and portability
* Kubernetes or Docker Swarm for container orchestration
* Ansible or Terraform for Infrastructure as Code
* Prometheus and Grafana for monitoring and alerting
* HashiCorp Vault for secrets management and secure access
* OAuth or OpenID Connect for authentication and authorization

Frontend:

* ReactJS for building web-based user interfaces
* Redux for state management
* Bootstrap or Material UI for styling and layout
* Axios or Fetch for making API requests to the backend

Database:

* PostgreSQL for data storage
* Cassandra or HBase for NoSQL data storage

The architecture of the system is given below:

Data Source (e.g. Kaggle dataset)

|

Data Collection Microservice (Python Flask)

|

Message Queue (Apache Kafka)

|

Data Processing Microservice (Apache Spark)

|

Data Validation Microservice (Python Flask)

|

Data Storage Microservice (PostgreSQL)

|

Machine Learning Microservice (Python Flask)

|

Frontend Application (ReactJS)

Using Docker and docker-compose (or similar) to containerize and orchestrate microservices is considered a recommended approach for building a reliable, scalable, and maintainable system. A platform for developing, shipping, and running applications is provided by Docker, while docker-compose makes it easy to manage and orchestrate multiple containers.

In our system, the following microservices will be containerized and orchestrated using Docker and docker-compose:

* The data collection microservice
* The data processing microservice
* The data validation microservice
* The data storage microservice
* The machine learning microservice

Separate Docker images will be created for each microservice, allowing them to be deployed and scaled independently. The containerization of these microservices will be managed using the docker-compose file, which specifies their dependencies and orchestrates their communication.

By containerizing each microservice, it can be ensured that each service is isolated and independent, making it easier to deploy, update, and scale. A consistent runtime environment is also provided by Docker, which helps to ensure that the application works the same way on any system, regardless of the underlying infrastructure. docker-compose will allow for the easy orchestration of these containers, enabling the management of the application as a whole.

## **2. Development phase/reflection phase**

The development phase is the stage where the data processing system is practically implemented. The followed stages are summarized below:

A new Git repository will be created to store all the code generated during this project. The directory structure will be organized to make sense for the project and the code will be accordingly organized.

data-processing-system/

├── data-collection-microservice/

│ ├── Dockerfile

│ ├── requirements.txt

│ ├── app.py

│ └── ...

├── data-processing-microservice/

│ ├── Dockerfile

│ ├── requirements.txt

│ ├── spark-job.py

│ └── ...

├── data-validation-microservice/

│ ├── Dockerfile

│ ├── requirements.txt

│ ├── app.py

│ └── ...

├── data-storage-microservice/

│ ├── Dockerfile

│ ├── requirements.txt

│ ├── init.sql

│ └── ...

├── machine-learning-microservice/

│ ├── Dockerfile

│ ├── requirements.txt

│ ├── app.py

│ └── ...

├── frontend-application/

│ ├── Dockerfile

│ ├── package.json

│ ├── public/

│ ├── src/

│ └── ...

├── docker-compose.yml

├── README.md

├── LICENSE

└── ...

1. The necessary microservices to process and analyze the data will be implemented using the design from the conception phase. Each microservice will be developed as a separate Docker container that can be deployed and scaled independently.
2. The Docker containers will be modified as necessary to include custom configurations and dependencies needed for the application.
3. The Docker containers will be deployed locally to the development machine for testing and debugging purposes. Each microservice will be tested and debugged to ensure that it is running as expected.
4. Throughout the development of the microservices, factors such as data security, governance, and protection will be considered to ensure that they are reliable, scalable, and maintainable.
5. The data chosen in the conception phase will be ingested into the system to ensure that it is ingested, pre-processed, and aggregated as expected.
6. The system will be tested and debugged to ensure that it is functioning as expected. Tools such as logs, monitoring, and performance testing will be used to identify any issues and address them.
7. Throughout the development phase, online meetings and other communication channels will be used to discuss ideas and drafts with colleagues and receive feedback. This feedback will be incorporated into the work to improve the quality of the final product.
8. Once the development phase is completed, the work will be handed in for evaluation. Feedback will be received from the tutor and any necessary revisions will be made.
9. Using the feedback from the tutor, work on the final draft of the data processing system will continue to ensure that it meets all the requirements and objectives outlined in the project brief.

## 

## **4.3 Finalization phase**

During the finalization phase, the code which builds the infrastructure is finetuned and the entire process is revised to identify areas that went smoothly and those that encountered problems. The system is assessed for reliability, scalability, maintainability, and technical requirements fulfillment. Measures to improve data security, governance, and protection are also considered, and a plan is discussed for introducing a second data pipeline capable of processing real-time streaming data.

Feedback is obtained through online meetings and other channels to improve the finished product before submission. The final product is accompanied by the results from Phases 1 and 2, as well as the required materials. An abstract is also mandatory, which describes the solution to the task in terms of content and concept and presents a brief technical approach breakdown in a sober and informative way.

Throughout the project, valuable technical skills are learned, and the major steps taken are discussed. The three most valuable technical skills learned during the project are identified, as well as the three most valuable "soft" skills.

Does your system fulfill the technical requirements?

Yes, the system should fulfill the technical requirements outlined in the project brief. During the finalization phase, it's important to review the requirements and confirm that all of them have been met. Any gaps should be identified and addressed before submission.

To ensure that the system meets the technical requirements, it's important to test and debug all components thoroughly. Use tools such as logs, monitoring, and performance testing to identify any issues and address them. Additionally, feedback from colleagues or other stakeholders can help to identify any potential gaps or areas for improvement.

What went wrong and why?

//Vom Kunde zu beantwortet, der das System implementiert

Is your system reliable, scalable, and maintainable?

Based on the development phase and testing, the system is reliable, scalable, and maintainable. The use of microservices architecture and containerization with Docker and Kubernetes makes the system scalable and portable. Additionally, monitoring and alerting with Prometheus and Grafana, as well as infrastructure as code with Ansible or Terraform, ensure that the system is maintainable and can be easily updated or modified. Furthermore, the use of Apache Kafka for real-time data streaming, Apache Spark for processing large datasets, and PostgreSQL for data storage make the system reliable in terms of data processing and management.

What measures for data security, governance and protection can be added?

There are several measures for data security, governance, and protection that can be added to a data processing system, including:

1. Authentication and authorization: Implementing a system to authenticate and authorize users to access certain data or perform certain actions can help to prevent unauthorized access and ensure data privacy.
2. Encryption: Encrypting sensitive data when it is stored or transmitted can help to protect it from unauthorized access.
3. Auditing: Implementing auditing capabilities that track user activity and system events can help to detect and prevent unauthorized access and data breaches.
4. Compliance with regulations: Ensuring that the system complies with relevant data protection regulations, such as GDPR or HIPAA, can help to protect against legal risks and potential fines.
5. Backup and disaster recovery: Implementing a backup and disaster recovery plan can help to ensure that data is not lost or compromised in the event of a system failure or data breach.
6. Monitoring and threat detection: Implementing a system to monitor and detect threats, such as intrusion detection systems, can help to detect and prevent data breaches.

What could you do in the next project to improve your workflow?

In the next project, there are several steps that can be taken to improve workflow:

1. Create a detailed project plan with clear timelines and milestones to keep the project on track and ensure that deadlines are met.
2. Use agile development methodologies to break down the project into smaller, more manageable tasks, and prioritize tasks based on their importance and urgency.
3. Continuously monitor and evaluate the progress of the project to identify any issues or areas for improvement.
4. Establish clear communication channels with team members to ensure that everyone is up to date with the progress of the project and can provide feedback and suggestions for improvement.
5. Use automation tools to streamline the development process and reduce manual errors.
6. Continuously test and debug the system to ensure that it is functioning as expected and identify any issues early on.
7. Stay up to date with the latest trends and technologies in the field to ensure that the project is using the best tools and methods available.

What are the major steps you took and what are the three most valuable technical skills you learned dur-ing the project?

Major steps:

* Conception phase: defining the project scope, objectives, requirements, and architecture.
* Development phase: setting up the Git repository, implementing the data processing infrastructure, modifying the Docker containers, deploying locally, ingesting data, testing and debugging, and using online meetings and feedback.
* Finalization phase: finetuning the code, revising the whole process, discussing strategies for introducing a second data pipeline, and submitting the finished product with an abstract and breakdown of the technical approach.

Valuable technical skills:

* Docker and docker-compose: containerization and orchestration of microservices.
* Python programming: implementation of data processing algorithms and machine learning models.
* Data pipeline development: ingestion, pre-processing, aggregation, and delivery of data using technologies such as Kafka, Spark, and SQL databases.

What are the three most valuable “soft” skills you learned during the project?

During this project, I learned several valuable soft skills, including:

* Communication: Communication is key when working on a team project, especially when it involves complex technical concepts. I learned how to effectively communicate my ideas and collaborate with my team members to achieve our common goals.
* Time management: Managing time effectively is essential when working on a project with strict deadlines. I learned how to prioritize tasks, plan ahead, and work efficiently to meet the project milestones.
* Adaptability: Throughout the project, we encountered various challenges and obstacles that required us to be flexible and adaptable in our approach. I learned how to quickly adjust to changes and find creative solutions to problems as they arose.

# **4. Conclusion**

In conclusion, this project aimed to develop a data processing system using microservices architecture and containerization with Docker. The goal was to build a reliable, scalable, and maintainable system that could ingest, process, and analyze large datasets. The project was divided into three phases: conception, development, and finalization.

In the conception phase, we defined the requirements and objectives of the project, selected the appropriate technologies and tools, and designed the architecture of the system. We chose to use microservices architecture with Docker containers and selected the necessary microservices to process and analyze the data. We also chose a relevant dataset that contained time-referenced data points and could be used for training and testing a machine learning model.

In the development phase, we implemented the data processing infrastructure using the design from the conception phase. We created the necessary microservices as separate Docker containers that could be deployed and scaled independently. We also modified the Docker containers to include custom configurations and dependencies needed for the application. We deployed the Docker containers locally for development purposes and tested and debugged each microservice to ensure that they were running as expected. We also ingested the data we chose in the conception phase into the system, pre-processed it, and aggregated it as expected.

In the finalization phase, we fine-tuned the code that built the infrastructure and revised the whole process, noting where things went smoothly and where problems emerged. We discussed the system's technical requirements, evaluated its reliability, scalability, and maintainability, and identified measures for data security, governance, and protection that could be added. We also identified areas for workflow improvement and listed the major steps we took and the most valuable technical and soft skills we learned during the project. Finally, we discussed a potential strategy for introducing a second data pipeline to the system that could process real-time streaming data.

Throughout the project, we used online meetings and other communication channels to discuss ideas and drafts with colleagues and receive feedback. This feedback was incorporated into the work to improve the quality of the final product. We also created a Git repository to store all the code generated during the project, which could be used as a portfolio and shown to potential future employers.

In summary, this project provided a hands-on experience in developing a data processing system using microservices architecture and containerization with Docker. It allowed us to gain valuable technical and soft skills and identify areas for workflow improvement. By using an iterative approach, incorporating feedback, and dividing the project into phases, we were able to build a reliable, scalable, and maintainable system that met the project's requirements and objectives.

# **References**

1. Kennedy K, Mac Namee B, Delany SJ, O’Sullivan M, Watson N. A window of opportunity: Assessing behavioural scoring. Expert Syst Appl. 2019; 40(4):1372–80.
2. Adams, M.N.: Perspectives on Data Mining. International Journal of Market Research 52(1), 11–19 (2018)
3. Asur, S., Huberman, B.A.: Predicting the Future with Social Media. In: ACM International Conference on Web Intelligence and Intelligent Agent Technology, vol. 1, pp. 492–499 (2018)
4. Bakshi, K.: Considerations for Big Data: Architecture and Approaches. In: Proceedings of the IEEE Aerospace Conference, pp. 1–7 (2012)
5. Cebr: Data equity, Unlocking the value of big data. in: SAS Reports, pp. 1–44 (2021)
6. Cohen, J., Dolan, B., Dunlap, M., Hellerstein, J.M., Welton, C.: MAD Skills: New Analysis Practices for Big Data. Proceedings of the ACM VLDB Endowment 2(2), 1481–1492 (2019)
7. Cuzzocrea, A., Song, I., Davis, K.C.: Analytics over Large-Scale Multidimensional Data: The Big Data Revolution! In: Proceedings of the ACM International Workshop on Data Warehousing and OLAP, pp. 101–104 (2021)
8. Economist Intelligence Unit: The Deciding Factor: Big Data & Decision Making. In: Capgemini Reports, pp. 1–24 (2012)
9. Sanchez, D., Martin-Bautista, M.J., Blanco, I., Torre, C.: Text Knowledge Mining: An Al-ternative to Text Data Mining. In: IEEE International Conference on Data Mining Work-shops, pp. 664–672 (2008)
10. Serrat, O.: Social Network Analysis. Knowledge Network Solutions 28, 1–4 (2019)
11. Shen, Z., Wei, J., Sundaresan, N., Ma, K.L.: Visual Analysis of Massive Web Session Da-ta. In: Large Data Analysis and Visualization (LDAV), pp. 65–72 (2014)