BEng Individual Project

Project Report

Integrated Development Environment for Automatic Deployment and Configuration of Big Data Applications

Big Data Auto-tuning Tool

July 2, 2017

Derek Law

Supervisor:
Dr Giuliano Casale

Contents

1 Abstract

Big Data applications are crucial in the data-driven economy, but the difficulty in configuring these systems for performance makes development inaccessible to less experienced programmers. This project aims to provides a fully integrated development environment built upon DICE tools to accelerate development of big data applications with a fully automated workflow, and integrate all the tools to help developers improve performance of big data applications built on popular technologies at ease.
The Big Data Auto-tuning Tool allows developers to select configuration parameters to be optimised, execute performance testing, and view retrieved results from the integrated development environment. The tool integrates with Jenkins automation server, and the extended BO4CO configuration optimisation tool, to deploy tests on big data applications automatically and remotely in the background.

2 Background

2.1 Motivation

Data is a crucial resource in all economic and social activities of our society, with an estimated 16 zettabytes of useful data (16 Trillion GB) [38] to be accumulated by 2020. Data sources are omnipresent in our lives, including streams from financial markets, video monitoring systems, social media streams and web activity logs. Exploiting this data to increase competitiveness in an increasingly data-driven socioeconomic model will boost innovation and drive cutting-edge technologies, which is key to economic growth. The demosEUROPA [33] estimated that 'Overall, by 2020, big & open data can improve the European GDP by 1.9%, an equivalent of one full year of economic growth in the EU', while the International Data Corporation (IDC) [1] shows that large companies and SMEs are accelerating their adoption of Big Data as they recognise the market value proposition and need to step forward into the data-driven economy.

Despite the great demand for Big Data, the development and deployment of Data Intensive Applications (DIAs) requires significant expertise and remains relatively inaccessible to new and less experienced programmers. The configuration of underlying technologies such as Hadoop, Spark, and Storm, has a significant impact on the performance of DIAs up to orders of magnitude [31]. This requires expert administrators with clear understanding of the technologies used, in order to test and tune the systems using a mix of trial-and- error and heuristic methods. Even with experienced administrators, manual configuration is virtually impossible due to the number of configuration parameters and possibly nonlinear interaction between them.

The Big Data Auto-Tuning tool is a solution to finding optimum configuration for DIAs, helping developers with little or no knowledge of the underlying big data technologies automatically improving their application's performance. The following sections further detail the current state of tools, which consists of separate parts including the BO4CO tool, test runners and monitoring tool. The main aim of this project is to provide an environment that:

- Fully integrates with DICE project development environment as an Eclipse IDE plugin
- Facilitates continuous integration by providing automated deployment and performance testing of big data applications
- Automatically detects optimal configuration to enhance big data application performance without requiring developer knowledge of underlying system

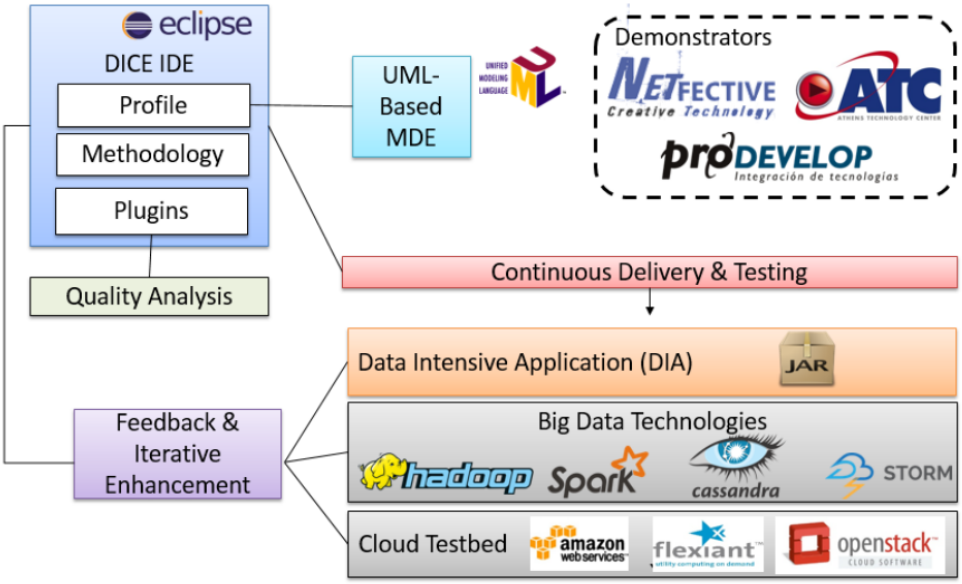
2.2 DICE Project

DICE is a project on quality-driven development of Big Data applications.[4] It researches methodology and tool chain to allow high quality, high performance Big Data applications to be developed with less experience and less domain-specific expertise required. The project covers quality assessment, architecture enhancement, continuous testing and agile delivery based on the DevOps paradigm.

The DICE framework is structured as in Figure 1, and divided into four main components [5]:

- DICE IDE: for coding, design and prototyping
- Quality Analysis Tools: by simulation, verification and optimization
- Feedback and Iterative Enhancement Tools: a resource, performance and anomaly monitoring platform tailored for Big Data technologies
- Continuous Delivery and Testing Tools: with continuous integration, delivery on clouds and optimal application configuration

Figure 1: DICE framework. [5]



2.2.1 DICE IDE

The DICE IDE is an integrated development environment built on the Eclipse IDE. It provides MDE functionality to model Big Data applications and the Big Data technologies behind. It is the front-end user interface for integration of all DICE tool chain components.

2.2.2 Quality Analysis Tools

The Quality Analysis Tools assess requirements and parameters specified in UML design for performance, reliability and safety. The Transformation Tools produce formal models, such as stochastic Petri nets, for validation and simulation in the Simulation Tools. The safety properties are checked using Verification Tools, then the Optimization Tools evaluate optimal deployment configuration using Petri net models.

2.2.3 Feedback and Iterative Enhancement Tools

The monitoring platform provides feedback by analysing traces and logs stored during execution of application on Big Data framework. Anomaly Trace Tools apply machine learning algorithms to enable the detection of quality incidents and anti-patterns based on classification, statistical analysis and trace analysis. Enhancement tools correlate monitor data to the application's design models, and guides quality improvement.

2.2.4 Continuous Delivery and Testing tools

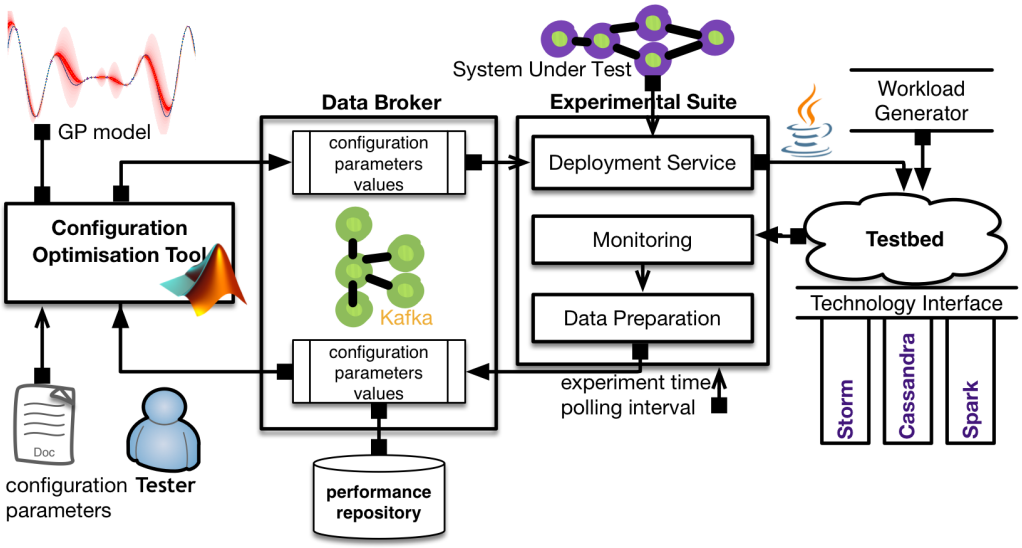
The Continuous Integration and Deployment tools set up the testbed and deploy the application to pre-production environment, where quantitative performance and reliability feedback on test input feeds is assessed as part of a DevOps work flow. The integration of automated deployment and testing tools internally and with existing development tools such as Eclipse and Jenkins will be key objectives in this project, to provide a seamless work flow that improves development efficiency.

2.3 Configuration Optimization Tool

Tuning the performance and reliability of Big Data applications is time consuming, as applications can involve several frameworks such as Apache Storm, Kafka, and Spark. The default parameters may not work optimally in different situations, and well-tuned systems can perform up to orders of magnitude better, making it crucial to find the optimal configuration [31]. Each framework has hundreds of configuration parameters that can present a few options or accept a range of values. There are $(num\_options) \wedge (num\_params)$ of possible configuration combinations, which easily reaches 1M possibilities when we choose to optimise 10 key parameters with 4 configuration options. A brute force approach that tested each possibility for 10 minutes of run time will take 19 years to find the optimal solution. The Configuration Optimization Tool has to find a good set of configuration parameters by experimentation within a limited time. A machine learning algorithm, BO4CO has been developed for this purpose, and is introduced in the next section. The Configuration Optimization Tool is structured as shown in Figure 2, and has the following components:

- Configuration Optimizer (BO4CO)
- Data Broker and performance repository
- Experimental Suite: for running tests on Testbed and monitoring

Figure 2: Configuration Optimization Tool. [32]



The configuration optimization process: [31]

1. Initialisation: Tester inputs parameters and possible values to test
2. Iteration 0: Working iteratively, select next configuration to test based on existing known performance measures

3. Deployment: DICE deployment tool automatically deploys the configured system on a testbed

4. Monitoring: DICE monitoring tool measures and stores the performance of the configuration to be added to historical data

5. Next Iteration: Select next configuration to test by fitting historical data to a model, go to step 3

6. Result: Terminate after the limited time has been reached, reporting the optimal configuration.

2.3.1 BO4CO Algorithm

BO4CO algorithm [31] is used by the Configuration Optimizer to automatically select the best configuration to test next. The Algorithm uses observed performance data to estimate the response surface of the Big Data system under testing, and uses the estimated data to search for points where there is a high probability the optimum configuration lies. The goal is

$$x* = argmin f(x)$$

, where

$$X = Dom(X1)x...xDom(Xd)$$

is the configuration space and

$$yi = f(xi)$$

is the partially known response function

1. Initialisation: form initial Latin Hypercube Design

2. Initial Measurement: form initial estimate of underlying response function from measurement on initial design

3. Repeatedly accumulate data with $yi = f(xi) + \epsilon i$ where ϵi is noise in measurement

4. Form GP $y = f(x) \sim GP(\mu(x), k(x, x'))$, where $k(x, x')$ is the distance between x and x'

5. Assuming $S1 : t = (x1 : t, y1 : t)|yi := f(xi)$ is a collection of t observations, the function values are drawn from

$$K = \begin{vmatrix} k(x1, x1) & ... & k(x1, xt) \\ ... & ... & ... \\ k(xt, x1) & ... & k(xt, xt) \end{vmatrix}$$

6. To fit a new GP model with observations accumulated so far:

$$\begin{vmatrix} f1 : t \\ ft + 1 \end{vmatrix} = \begin{vmatrix} K + \sigma^2 I & k \\ k^T & k(xt + 1, xt + 1) \end{vmatrix},$$

where $k(x)^T = [k(x, x1)k(x, x2)...k(x, xt)]$ and I is identity matrix

7. new GP model created:

$$PR(ft+1|S1:t, xt+1) = N(\mu t(xt+1), \sigma^2(xt+1))$$

,
where

$$\mu t(x) = \mu(x) + k(x)^T(K + \sigma^2 I)^{-}1(y - \mu)$$

$$\sigma^2 t(x) = k(x,x) + \sigma^2 I - k(x)^T(K + \sigma^2 I)^{-}1k(x)$$

8. Select next configuration $xt+1$ to best tested:

$$xt + 1 = argmax\mu(x|M, S1:t)$$

,
where the criteria μ uses Lower Confidence Bound, which trades-off between exploitation and exploration:

$$\mu LCB(x|M, S1:n) = argmin\mu t(x) - k\sigma t(x)$$

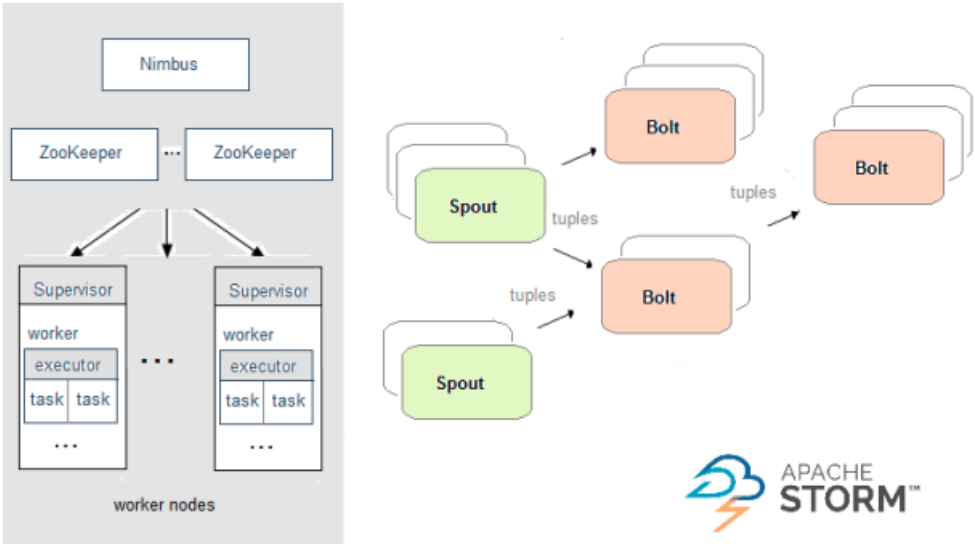
2.4 Big Data Frameworks

Big Data Frameworks or Big Data Platforms are the modern solution for storing and processing Big Data in the Petabyte and Exabyte scales. While tradition monolithic systems that scale vertically face physical limitations in their power, parallel distributed storage and processing systems that scale horizontally do not face the same limitation and can work in hundreds or thousands of nodes. These systems are controlled by new frameworks including Hadoop, Spark and Storm. This project focuses tailoring performance in Spark and Storm applications by recognising key configuration parameters and building specific profile templates through experimentation.

2.4.1 Storm

Storm shares the topology shape of a directed acyclic graph, similar to MapReduce. [35] The data transformation pipeline consists of spouts and bolts at the graph vertices, and named streams directing data across nodes on the edges. The difference in Storm and MapReduce is that Storm processes data in real time, making it easy to process unbounded streams. Storm topologies run indefinitely until killed, while MapReduce jobs will terminate and are processed in batches. The benefits that Storm offer is that it can process large volumes of high velocity data, up to one million 100 byte messages per second per node. It is scalable, fault tolerant and easy to operate. The guarantee that every message is processed makes it reliable. Use cases include real time analytics, online machine learning, continuous computation, distributed RPC and ETL.

Figure 3: Storm architecture. [17]



2.5 DevOps

2.5.1 Introduction to DevOps

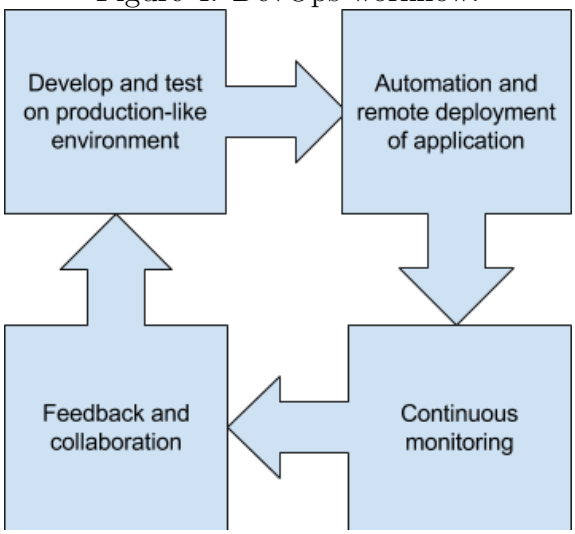
The term DevOps originated from "software DEVelopment" and "information technology OPerationS". DevOps is the combination of cultural philosophies, practices, and tools that aims to increase the speed, frequency and reliability in building, testing, and releasing software.

Co-operation, collaboration and communication between software developers and Operations teams is the key difference championed in the DevOps paradigm, as opposed to the traditional separation of concerns. Before the rise of DevOps, developers in the "Dev" side were seen as the "creators", focused on innovating, implementing new features and improving the quality of the product. The "Ops" side involved system administration and maintenance, focused on "everything after the creation of the product", ensuring stability, availability and reliability. This rigid structuring of responsibilities could lead to friction as both roles affect the work of each other, and would instead benefit from the insights of the other in a more flexible system.

The key ideas of DevOps are as shown in Figure 4 [39]:

- Software testing in an environment as close as possible to production environment, encouraging developers to consider operation issues including performance, availability and stability of application
- Automation and remote deployment of application
- Continuous monitoring of application throughout entire life-cycle
- Feedback and collaboration between developers and operators to resolve issues and improve application considering the interest of both parties

Figure 4: DevOps workflow.



These practices aim to bridge the gap between developers and operators, encouraging them both to consider the perspective of the other, understand the basic of

others domains, and share ideas, goals, issues, processes and tools [16]. In this sense, DevOps is an extension to Agile practices that encourage quick iterations that deliver incremental improvements in the product. DevOps extends the idea across developers and operators, pushing both to quickly and frequently deliver product improvements [16].

In addition to communication, integration and sharing of tools also help facilitate co-operation and collaboration between developers and operators [20]. Tools enable the automation of building, testing, releasing and deploying applications, while ensuring that the common process is reproducible and identical for developers and operators alike.

2.5.2 Big Data Auto-Tuning Tool and DevOps

The Big Data Auto-Tuning Tool's positioning within the DevOps paradigm is as an integrated tool that facilitates performance testing, monitoring, and feedback, by providing automated deployment and configuration services to developers. It provides an intuitive user interface that is fully integrated into the DICE development environment on the Eclipse IDE. Through Jenkins and Chef, commonly used DevOps tools, it automatically tests Big Data applications in a production environment, and reports performance metrics and optimal configuration detected. This allows developers with little knowledge of Big Data technologies to consider the performance aspect of their application, in line with the first key idea of DevOps. As part of the DICE project, it is focused more on development and initial prototyping phases, and less on managing the application on the operations side.

2.6 Jenkins Continuous Integration

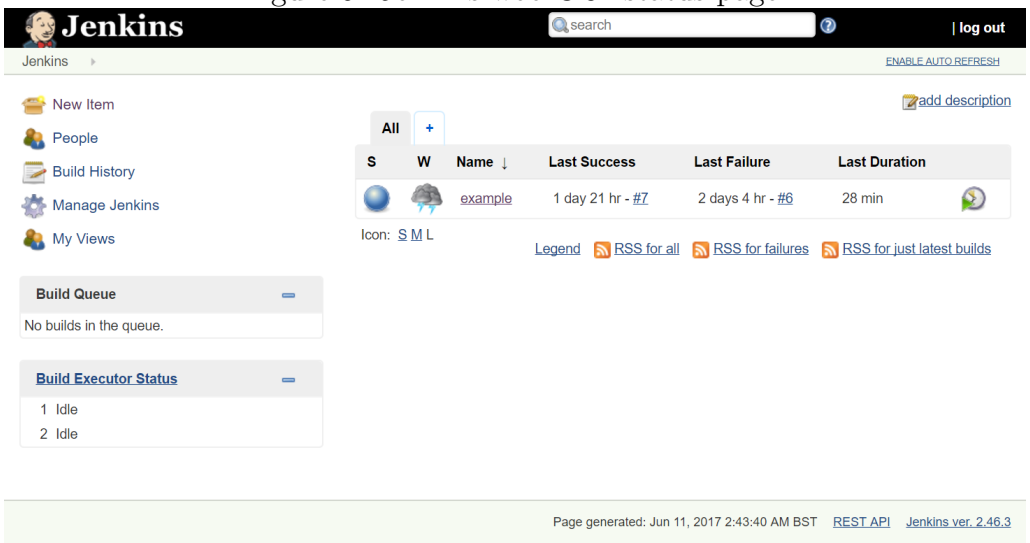
Jenkins is an automation tool commonly used in continuous integration and continuous delivery. As such it is one of the common tools in the DevOps toolchain. It shares its roots with Oracle's Hudson project, [23] and was designed to help build, test and deploy software.

2.6.1 Using Jenkins

Jenkins runs as a server that can be installed onto the system and application containers, or as standalone Java process. It is availale for use with Windows, Mac OS, various Linux systems and Docker application containers. As a generic Java package (.war), it can also be used with various servlet containers such as Apache Tomcat.

As show in Figure 5, Jenkins provides a web GUI where users can configure the

Figure 5: Jenkins web GUI status page.



system, create automation jobs and monitor the status of jobs. A job is called a "project", and is commonly a continuous integration cycle of building an application from source code and executing tests for the application. The main page of the Jenkins web GUI shows a list of all defined projects, and statistics about the status of the project. A project in Jenkins always terminates with one of two outcomes: success or failure. A project in Jenkins is defined by a set of user defined rules. These rules are usually one of the following categories:

- Source Code management - Jenkins can be configured to retrieve source code from version control tools including Git and Subversion.
- Build trigger - by default all jobs can be started manually from the web GUI. It can also be configured to be started by remote api calls, or by user defined triggers such as the end of another Jenkins job, at user defined intervals, or when the source code is detected to have been updated.

- Build steps - Jenkins can build Apache Maven based projects, and/or execute any user defined shell scripts and Windows batch commands.
- After build actions - Jenkins can be configured to store build artifacts, store reports, and trigger other builds.

The generality of build steps that can involve any user defined shell script makes it possible to use Jenkins for automating any task that is not limited to building and testing applications.
Jenkins projects are also dynamic, and a set of parameters can be passed to the project when it is started. Users can define names and types of parameters, and the values are to be set whenever the project is triggered. The parameters will be available as environment variables throughout the project cycle. The types of parameters are extensible, with support the following types built-in: [24]

- Boolean
- Choice - Categorical options
- File - file to be uploaded to the workspace
- String / Multi-line String / Password - various text values
- Run - URL of a specific run of another Jenkins project

2.6.2 Jenkins Plugins

Jenkins is open source and provides extensibility through plugins. The plugins provide extra functions, powerful ways to define projects, and integrations with many testing and deployment technologies. Various development environments are supported through plugins, including: [22]

- Apache Ant projects
- Gradle projects
- .Net projects - MSBuild and MSTest
- Java projects - JUnit and built-in support for Maven
- Android - Android emulator
- iOS - Xcode

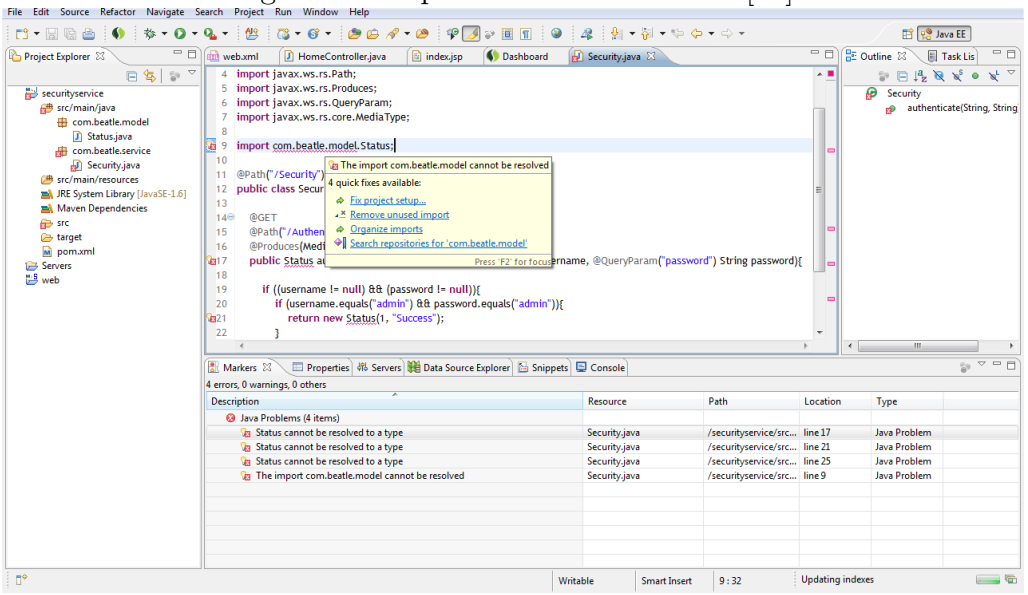
Developers working with these environments have an established framework to help set up their Jenkins automation routine of building and testing. Jenkins also allows manual configuration to build and test applications developed in other environments and languages, through the ability to execute any user defined shell script. This process requires more customisation and knowledge from the developer to understand build and testing behaviour.

2.7 Eclipse IDE Plugins

2.7.1 Eclipse IDE

Eclipse is a commonly used Integrated Development Environment for Java developers, along with a significant userbase among C/C++ and PHP developers. [7] It is the developement environment that the DICE project has targeted, and built the DICE IDE upon. [5]
The architecture of Eclipse IDE is based on the principles of modular systems and

Figure 6: Eclipse IDE user interface. [21]



services, for which the OSGi core framework has set a specification [30]. Equinox is an implementation of the framework, and forms the basis of the Eclipse IDE's core runtime system kernel. Eclipse has a concept of a workspace, and all functionality is provided by plugins on top of Equinox, as modular components called bundles. According to the OSGi architecture, bundles are tightly coupled, dynamically loadable collections of classes, jars, and configuration files that explicitly declare any of their external dependencies. This allows all functionlity and plugins in Eclipse to integrate into the system in the same way.
With this architecture, the Eclipse IDE allows extensive customisation and easy extension support developement in different languages, and add fucntions such as configuration management and version control. Development environments are provided for the following languages: [9]

- Java - Eclipse Java development tools (JDT), with advanced refactoring techniques and code analysis
- C and C++ - Eclipse C development tools (CDT)
- PHP - Eclipse PHP development tools (PDT)
- Android - Eclipse for Android Developers

- Javascript - Eclipse IDE for JavaScript and Web Developers

Development for the following programming languages are supported via plugins: [8]

- Ada
- ABAP
- Clojure
- COBOL
- D
- Erlang
- Fortran
- Groovy
- Haskell
- Julia
- Lasso
- Lua
- NATURAL
- Perl
- Prolog
- Python
- R
- Ruby, Ruby on Rails
- Rust
- Scala
- Scheme
- TeXlipse - development of LaTeX documents

Other plugins in eclipse provide integrations and additional functionality: [8]

- EGit - integration of Git version control system
- Subversive - integrationi of Subversion (SVN) version control system
- Buildship Gradle Integration
- m2e - Maven integration
- Checkstyle - source code analyzer Checkstyle.
- Eclipse Memory Analyzer - Java heap analyzer

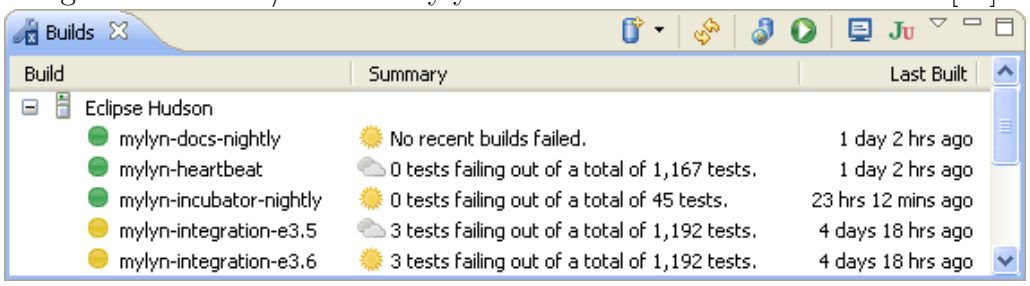
The Eclipse UI is built upon the Standard Widget Toolkit (SWT, a Java toolkit that contains graphical control elements. It also uses JFace as the intermediate graphical user interface layer to simplify application construction.

2.7.2 Eclipse and Jenkins

Following the DevOps principles of testing in production environment, automation and remote deployment of application, we investigate the current state of integration in the development environment (Eclipse IDE) and the build automation tool (Jenkins). The current integration is provided via the "Hudson/Jenkins Mylyn Builds Connector" under the Mylyn task and application lifecycle management (ALM) framework for Eclipse. [37]
The plugin describes itself as a "View to review, start builds in CI server", and lists the following features:

- connect to many Hudson/Jenkins instances, filtered list of jobs
- view build logs in Eclipse, error stacks are clickable to open classes

Figure 7: Hudson/Jenkins Mylyn Builds Connector user interface. [37]



- view JUnit results, and rerun them locally

Figure 7 shows the user interface of the Hudson/Jenkins Mylyn Builds Connector. It mimics the summary view of the main Jenkins web GUI landing page shown in Figure 5. This demonstrates the ability to log in to a secure Jenkins server with the provided credentials, and retrieve the status of projects. The ability to start a build has been investigated, and it was found that the plugin is only capable of starting basic builds that did not require any parameters. Parameterized builds were not supported by the plugin, and it also did not provide the functionality to create and define new project builds.

2.7.3 Eclipse Plugin Development

As described in the sections above, all functionality in Eclipse are provided via plugins. The Eclipse IDE can be extensively extended with any functionality by developing plugins on top of the platform. This is an implementation of the OSGi framework and its principles: to have a modular system; where eclipse plugins are implementations of OSGi bundles.
Even the Eclipse IDE itself can be seen as an extension and implementation of the Eclipse RCP, with the function to aid software development. Although released as a whole package, these software development functionalities of the IDE are actually sets of plugins to the system, such as the Java Development Tools and C development tools. When these plugins are uninstalled, Eclipse IDE is still capable of launching as an application without any development support functionality.
Eclipse for RCP and RAP Developers is the package that includes a set of plugins to form the Eclipse plugin development environment (PDE). These tools provide the following functionalities: [18]

- project creation wizard - configure and build plugins from style templates
- dependency management - Manifest
- launch, test and debug - launch a new eclipse instance with plugin built and installed for testing

When a plugin project is started from the project creation wizard, the project directory structure is configured automatically, along with plugin dependencies, metadata and the manifest file. The developer manually create a custom plugin from scratch,

or select from the following types of plugin templates to suit the style and purpose of their plugin:

- Plugin with a multi-page editor
- Plugin with a popup menu
- Plugin with a property page
- Plugin with a view
- Plugin with an Eclipse 4 handler
- Plugin with an Eclipse 4 view
- Plugin with an editor
- Plugin with an incremental project build
- Plugin with sample help content

Each of the templates provide the UI's top level display style as a blank canvas with some sample control buttons for demonstration purpose. Developers can add their own controls to the top level display according to the SWT toolkit. In the SWT toolkit, all UI elements such as the display canvas, text, and buttons, are controls that follow a tree type hierarchy from the top level display provided by the template.

2.7.4 SWT: The Standard Widget Toolkit

The Standard Widget Toolkit (SWT) provides Java developers with widgets that tightly integrate with the underlying native OS GUI platform, with an API that is portable across various platforms. The API across platforms is common, but implemented with platform native widgets when they are available. Therefore SWT reflects the native look and feel of the underlying OS GUI [14], and any changes to its style. The native OS's GUI libraries is accessed via Java Native Interface in order to display GUI elements. In this sense, SWT is similar to programs written using APIs specific to each operating system.
As SWT uses native objects that are not tracked by the Java JVM, the JVM's garbage collection mechanism cannot track the usage of these objects, and cannot automate the garbage collection of SWT widgets. Therefore SWT is different from any other Java toolkit in that it requires manual deallocation of objects created. The deallocation is invoked by a call to the dispose method implemented by all SWT widgets. [15] Each SWT widget handles the disposal of the native OS objects or delegates disposal to lower level widgets by invoking their dispose method.

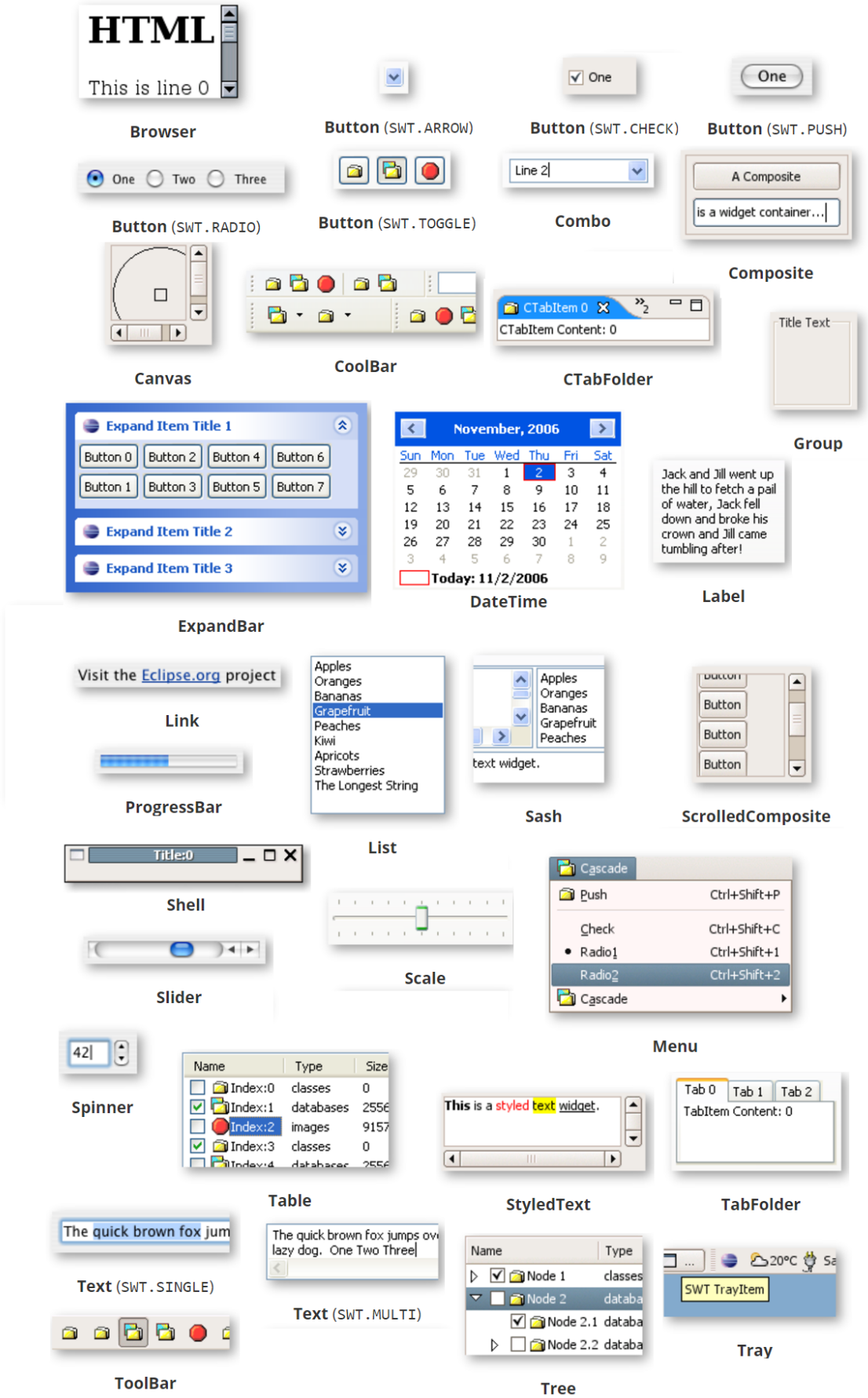
The core architecture of SWT is formed by widgets, layouts and events. As a comparison to the MVC model, widgets are graphical objects that also contain data, layouts are specifications of GUI style, and events hold the logic and actions in response to user interaction.
The main execution entry point of the UI is the Display. It manages communication

with the OS's GUI system, and coordinates the event loop. [15]
The Display will contain top level Shells as its children. Shells are the visually represented as "windows" by the underlying OS. They can be moved, resized, minimized and maximized, and may contain shells within shells, for example pop-ups and dialogue boxes that are entirely "contained" by another window.
This concept of "containing" gives rise to the widget's hierarchical structure. The shell / application window is at the top of the hierarchy, as the root of a tree. The shell will have other widgets as children, and those widgets may in turn contain more widgets further down the tree levels. As a rule, all widgets that are not top level shells must have a parent. Top level shells do not have a parent, but they are created in association with a particular Display. All other widgets are created as descendants (direct or indirect) of top level shells.
The Control class is a general abstract class of widgets that can be placed at any level of the hierarchical tree, while the Composite class are widgets that can contain other widgets, i.e. can have children. [6]
Figure 8 shows screen shots of SWT widgets, while the detailed description can be found in the appendix.

The appearance and positioning of widgets can be modified by style bits and layout. [11] Style bits set in the constructor of widgets determine properties such as whether widget has border outline, and whether the widget has scroll bars. A widget may have many style properties, represented by the bitwise OR of all style bits. When the selected style is not available on a platform, it is ignored.

The event and listener pattern is used to control logic and respond to user interaction in SWT. Messages are passed to widgets from the application's message loop, containing information about events from the queue. Widgets then notify their listeners for the specific event, and the listener contains logic to respond to the event. Listener interfaces are defined for different types of listeners, to specific what trigger events they should react to. For example, a "open" button's selection listener will listen to selection events, i.e. when the button is pressed, and execute the related code for the "open" action as defined by the developer.
Events are divided into two categories, where low level events describe specific user interactions, and high level events are logical operations or controls that may be formed of multiple low level events. [10] The details of SWT events can be found in the appendix.

Figure 8: SWT Widgets [13]



3 Project Implementation

3.1 Deliverables

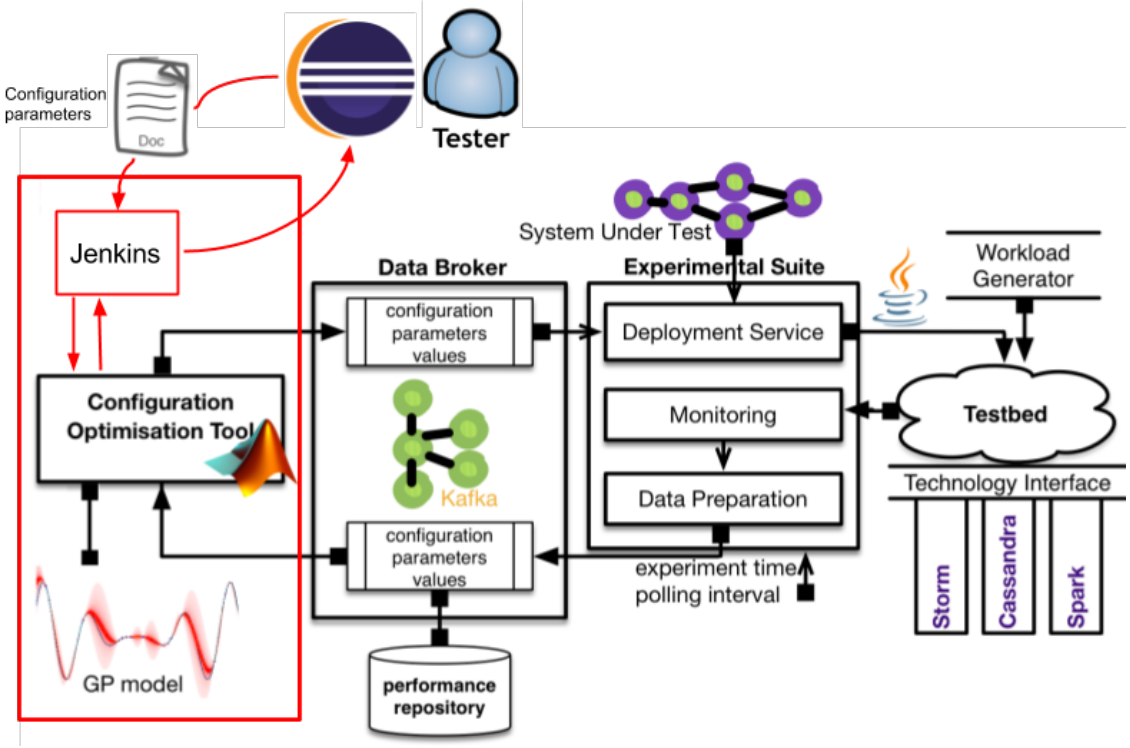
The Big Data Auto-Tuning tool is a solution to finding optimum configuration for big data applications, helping developers with little or no knowledge of the underlying big data technologies automatically improving their application's performance.
The implementation of the tool has two main parts, which consist of the following deliverables:

- - Provide a development environment in the Eclipse IDE, where developers can explore configuration optimisation and the performance of their applications.
  - Allow selection of configuration parameters of corresponding big data technology for optimisation
  - Allow specification of parameter values, ranges and intervals to experiment upon
  - Extend BO4CO tool to support broader types of input: boolean, categorical, ranges and intervals.
  - Allow configuration of experiment set-up, eg: test application to run, numbers of iterations and experiment time
  - Allow setting of connections to remote Jenkins server, remote testbed and monitoring services
- - Fully integrate the development environment to trigger configuration optimisation on remote automation server and run tests on remote testbed.
  - Integrate Eclipse and Jenkins for triggering parameterised builds with configuration files remotely
  - Integrate Jenkins automation server and MATLAB-based BO4CO tool to start experiment
  - Integrate BO4CO tool and remote Storm testbed to deploy tests with different configuration parameters and retrieve performance metrics
  - Integrate Eclipse and Jenkins to retrieve and display BO4CO configuration results

3.2 Big Data Auto-tuning Tool Design

The Big Data Auto-tuning Tool builds upon the structure of the Configuration Optimization tool, detailed in Figure 2. The tool adds two main parts, a user interface on the Eclipse integrated development environment, and a Jenkins server instance that automates execution of the Configuration Optimisation tool. As seen in Figure

Figure 9: Big Data Auto-tuning Tool.



9, the developer selects configuration parameters to optimise via the user friendly interface on the Eclipse IDE. The experiment configuration file is generated according to the selections, and sent to the remote Jenkins CI server as it triggers the experiment to run. Jenkins executes the Configuration Optimisation tool, and monitors the status of the experiment. When the CO tool terminates, Jenkins retrieves the results and the optimised configuration. The developer can access the results conveniently from the Eclipse IDE.

The Big Data Auto-tuning Tool provides a fully integrated solution in the development environment to performance test and tune big data applications, in line with the DevOps principles of frequent testing in deployment environment. In comparison to the existing approach to directly work with the Configuration Optimisation tool, the developer can now run optimisation and view results without leaving the IDE. The developer no longer manually creates an experiment configuration file for the tool. The GUI lists the configuration parameters available for selection for the corresponding big data framework, provided helpful descriptions to aid non-expert developers. It simplifies the process for the developer, without requiring that he understands the format of the CO tool's experiment configuration file. It also eliminates human error in creating the file and guarantees no parsing problem when it is

executed. The Big Data Auto-tuning Tool allows the use of a remote Jenkins server to execute experiments. The developer no longer has to have terminal access to the machine hosting the CO tool to run and monitor experiments. Jenkins remotely runs and retrieves results, leaving the developer free on his machine. The results are retrieved and clearly displayed, again requiring no access outside of the IDE.

3.3 Project Management and Risks

3.3.1 Original Timetable

Month	Milestone
March '17	Investigate and implement the integration of Eclipse and Jenkins
April '17	Create UI in the Eclipse IDE to select configuration parameters Complete UI in the Eclipse IDE to collect experiment settings
May '17	Set up Jenkins automation for execution of BO4CO with integration to testbed. Retrieve results to Eclipse and display optimal configuration.

3.3.2 Risks

There are significant risks in implementing the Big Data Auto-tuning tool, due to the complexity of coordinating the numerous components involved. The integration of all parts is the main challenge of this project, and the reward is that developers using the tool will have a smooth and integrated experience without the hassle of manually setting up each component to interface with each other. Some components reside on different remote servers and require connections to be established, and the components are built on different technologies and programming languages.
The integration between Eclipse and Jenkins was the first risk. The state of the art integration is provided by the Hudson/Jenkins Mylyn Builds Connector [37], which does not support triggering parameterised builds on the Jenkins server from Eclipse. Jenkins provides an API for triggering parameterised builds, but the specific requirements to include a file parameter was not documented [24], although an example using cURL was given [25]. It remained unclear what the general requirement was, and was unclear how to replicate the command in a Java / Apache-based HTTP client. As the configuration file parameter is crucial to this project, this was the first risk to tackle. A fall back option was also planned to parse an entire file into a String and trigger the parameterised build with a String parameter, which was a well-documented approach. [24]
The dependency of this project on core BO4CO code poses a big risk. It has to be assumed that BO4CO code provided was working as designed. The extension of BO4CO functionality to accept different input types to support categorical values in addition to integer values is also challenging because it requires changing the data type used internally, and the changes propaginate to the entire system that is tightly coupled.
The execution of experiments for configuration optimisation created a lot of dependencies on the big data frameworks installed on the testbed. The state of each of the following services each posed a risk to the successful integration of this project. If any of these services did not behave as they are designed to, or if there is any update that introduced incompatibilities, the project would be at risk of failing.

- DICE Jenkins Continuous Integration service - for monitoring experiments
- DICE Deployment Service - for deploying experiments
- DICE Monitoring Platform - to monitor performance of experiments

- Storm cluster - for executing experiments
- Zookeeper cluster - for deploying experiments
- Kafka cluster - for deploying experiments
- Hadoop cluster - for executing experiments

The integration between Jenkins and the MATLAB-based BO4CO code is also a risk. Unlike projects in Java or those managed by Gradle, Ant and Maven, MATLAB projects are not supported by a Jenkins plugin. The execution will have to be controlled by executing scripts, which make the implementation dependent and susceptible to changes in the host platform. There is an unofficial guide on MATLAB continuous integration using Jenkins written for Linux users, but there is no support for developers on Windows machines.

3.3.3 Project Time line

The initial stages of the project proceeded as planned in the original timetable. However, as mentioned in the risks above, the execution of BO4CO proved to be problematic. The problems encountered are documented in the later sections on BO4CO and integration with Jenkins and the testbed.
These difficulties caused the completion of the project deliverables to be delayed into June. In the interest of delivering a fully integrated development environment as specified in the deliverables, time and effort was dedicated to fully integrating all services to support Storm big data framework. Validation for use with the Hadoop framework was not completed, although support for the Hadoop framework had been built in.

Month	Progress
March '17	Integrate Eclipse and Jenkins for triggering parameterised builds with configuration files remotely.
April '17	Create UI to allow selection of configuration parameters. Allow specification of parameter values, ranges and intervals to experiment upon. Collection of experiment settings, connections to remote Jenkins server remote testbed and monitoring services.
May '17	Refine and complete Eclipse UI. Integrate BO4CO tool and remote Storm testbed to deploy tests with different configuration parameters and retrieve performance metrics. Integrate Eclipse and Jenkins to retrieve and display BO4CO configuration results.
June '17	Integrate Jenkins automation server and MATLAB-based BO4CO tool to start experiment. Extend BO4CO tool to support broader types of input: boolean, categorical, ranges and intervals.

3.4 Eclipse Plugin

3.4.1 Specification and Design

This section details the implementation to provide a fully integrated development environment in the Eclipse IDE, where developers can explore configuration optimisation and the performance of their applications. From the project deliverables, the following specification is defined:

- Allow selection of configuration parameters for optimisation - read a list of available parameters for different big data frameworks and dispaly for user selection
- Depending on the type of selected parameter, its values, ranges and intervals for the experiment to consider can be specified
- Allow configuration of experiment set-up, connections to remote Jenkins server, remote testbed and monitoring services. Generate formatted experiment configuration file to send.
- Integrate Eclipse and Jenkins for triggering parameterised builds with configuration files remotely
- Integrate Eclipse and Jenkins to retrieve and display BO4CO configuration results

The main components that form the Eclipse Plugin with the above functionality include:

- Parser to read available big data framework parameters
- Eclipse View as the main user interface to display and select parameters
- Configuration file generator to output experiment configuration
- Connector to Jenkins to trigger builds and retrieve results

The design outlines a user interface with different tabs for parameter selection, services configuration, experiement configuration, application configuration, and experiment execution. The parameter selection tab would show a list of available parameters, filtered by the specific big data framework. Developers can select relevant parameters from the list, and specify ranges or a set of values to test.
Parameters may be one of the following types: Integer, Boolean, or Categorical (String). The user interface must be capable of displaying differnt types of parameters with different detailed fields coherently.
The required output in the configuration file must follow a strict format to be accepted by the Configuration Optimization tool. The file is in the YAML format, and an example is provided in the link [3].

3.4.2 Configuration Parameters

The Big Data Auto-tuning tool supports four types of configuration parameters:

- Integer - the parameter value may be any integer between the LOWER BOUND and UPPER BOUND, with a user-specified STEP value that hints to the Configuration Optimising tool the minimum difference between each value to test.
- Percentage - the parameter value may be any percentage between 0 and 1, with a user-specified STEP value that hints to the Configuration Optimising tool the minimum difference between each value to test.
- Boolean - the parameter value may be true or false
- Categorical - the parameter may take any one of the String values from the OPTIONS list

In addition to the type specific fields above, all parameters have the following information:

- name
- list of applicable big data frameworks
- default value
- description

The initial implementation had one Parameter class with a TYPE field to distinguish between differnt types of parameters. This was refactorised to the final implementation of a Parameter interface with different implementing classes for each type of parameter. The design choice is discussed in the evaluation section.
Parameters available for selection are stored in the *params.xml* file. The XML schema of the file mimics the Java object representation. There are four types of elements for the four types of parameters, with each information field in the parameters stored as a child element in XML. The XML schema can be found in the appendix.
The current *params.xml* file contains a selection of configuration parameters from the Apache Hadoop and Apache Storm big data frameworks. These are the two main frameworks supported by the Configuration Optimization tool currently. The configuration parameters included in the file are selected from the officially documented configuration files of the respective frameworks. [19] [34] The aim is to select *performance related* parameters from the hundreds of parameters. Hence the selection process is based on removing obvious *non-performance related* parameters, such as parameters that:

- Specifies connections, external inputs and outputs. Eg: file path, service ports
- Controls security settings, Eg: usernames and passwords
- Control external functionality. Eg: logging

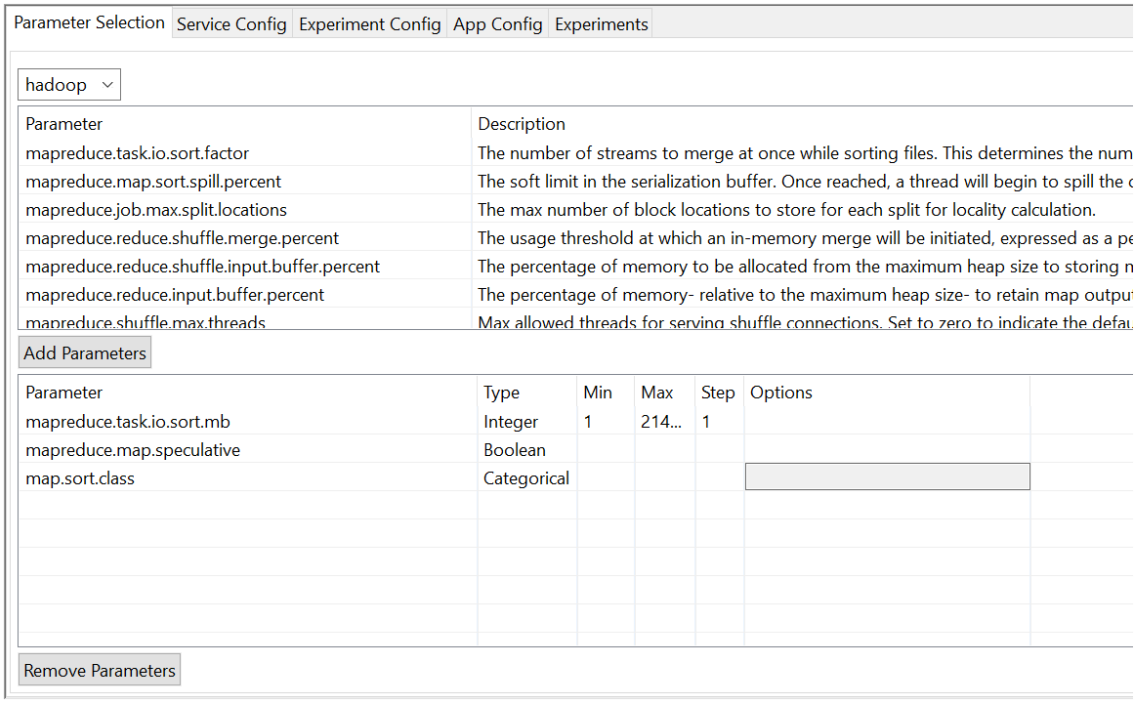
The *params.xml* file can be easily modified to reflect any changes in the parameters of big data frameworks. The file is read every time the Eclipse plugin starts, and any changes to the file will not take effect until the plugin restarts. Because it is data of all available parameters, the plugin never modifies its contents, and treats the file as read-only. The plugin creates a collection of parameter objects internally to temporarily store the parameter information for this session.

3.4.3 User Interface: Parameter Selection

The parameter selection tab is the most complex and most important tab in the Big Data Auto-tuning tool.

As shown in Figure 13, two tables form the main part of this tab. The following

Figure 10: Screen shot of parameter selection tab.



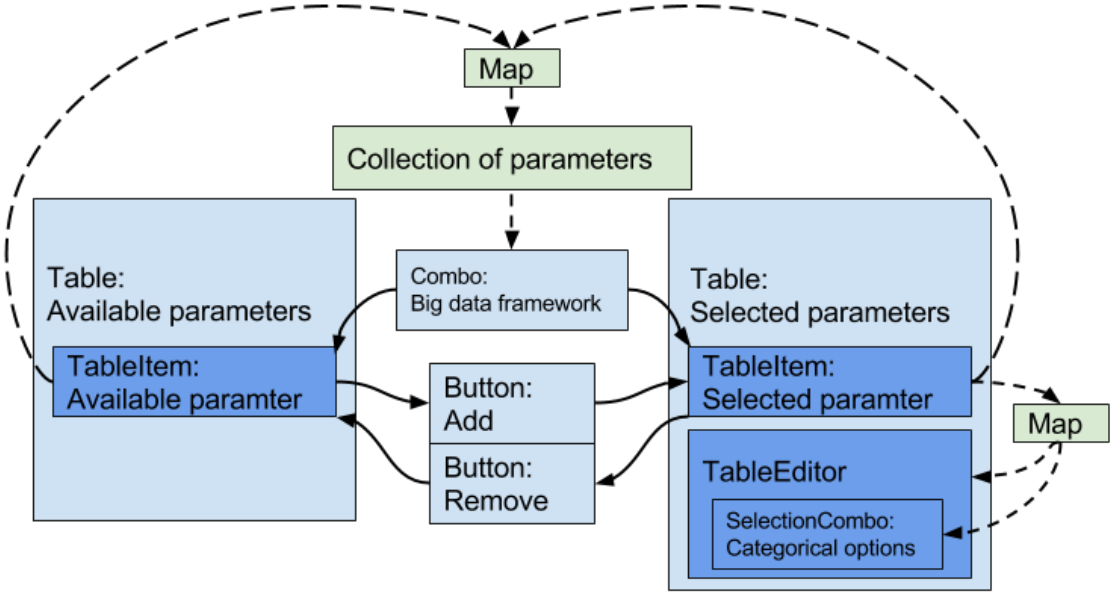
are the features of the Parameter Selection tab:

- Upper table displays all parameters available for selection, filtered by the big data framework they are applicable to
- A second column shows helpful description to explain the purpose of each parameter.
- The filtering is controlled with the drop-down list on the top left.
- Parameters available to select can be selected by:
  - Double-clicking anywhere on its row
  - Clicking the row and pressing the “Add Parameter” button
  - Multiple parameters can be chosen with common keyboard shortcuts such as “Ctrl + A”, “Ctrl + ArrowKey”, and “Shift + ArrowKey”.
- Lower table holds the selected parameters
- Lower table allows the developer to specify values to test each parameter with, instead of showing the description text

The challenge is to allow the developer to specify an arbitrary numerical range for integer and percentage type parameters, and choose from the available options of categorical parameters. There is no need to further restrict the values for Boolean parameters because the choice is binary. The challenge is to display and edit all the different types of details coherently in one table. All text in the table can be edited by double clicking the cell. While the table format is well-suited to taking pure texted based user input, such as the MIN, MAX, and STEP, for numerical inputs, it lacks the built-in ability to display a list of possible options and allow selection of categorical parameter values. A customized drop-down list is created to provide the functionality.

Figure 11 shows how the different components in the parameter selection tab are

Figure 11: Diagram showing parameter selection tab control and logic.



connected.

- Blue components are SWT widgets that appear on the user interface
- Green components are underlying data collections and objects
- Deeper shade of blue is used to highlight that multiple instance of the component may exist
- A SWT widget is the parent of SWT widgets contained inside its box on the diagram

When the plugin is launched, a collection of parameter objects is created from reading the *params.xml* file. The SWT Table [13] of available parameters is created, along with the drop-down list (SWT Combo [13]) that controls which parameters are to be displayed for the selected big data framework. The default selection is for the big data framework first in alphanumeric order. The table is populate with

SWT TableItem objects [13] that appear as rows on the table. These TableItems are children widgets to the table, and they carry a parameter name and description. Each TableItem is uniquely mapped to the Parameter object that it has been created to display.
The SWT Table is a powerful widget, as it contains internally all the logic to detect which items in the table are *selected*. The *selection* may be by mouse click, or by keyboard shortcuts mentioned above.
The SWT Table of selected paramters is created to be empty, along with SWT Buttons for "Add parameter" and "Remove parameter". These buttons will control the movement of parameters across the two tables. Each button has a listener, that listens for a MouseDown event on the button. The listener's MouseDown method is called when the event occurs, so we define the desired action by overriding this method.
When the "Add parameter" button is pressed:

1. Query the Table for available parameters for a list of *selected* parameter TableItems

2. Retrieve Parameter objects from mapping of TableItems to Parameter objects

3. Remove *selected* parameter TableItems from the Table for available parameters

4. Create TableItem in the Table for selected parameters for each Parameter object

5. Create a TableEditor to hold the SelectionCombo [29] with corresponding options for each Parameter object of categorical type, and map the TableItem to the SelectionCombo

A double-click listener was added to the Table of available parameters, with the same action of the "Add parameter" button, to enable more convenient user selection of parameters.
The categorical parameter is the most challenging parameter to display. It requires an arbitrarily long list of options to be shown, but the SWT Table format only supports text in cells of a predefined size. For a coherent user interface, the solution is to use a drop-down list style selection widget that minimizes to a fixed size of a cell. Multiple values from the available options of a categorical parameter has to be selected for experiementation, yet the SWT Combo widget does not allow more than one option to be selected. Therefore a customised widget was created for this purpose. The TableEditor [13] allows advanced widgets to be displayed in a cell in a Table, and dynamically manages the display position in relation to a TableItem. However, there is no direct link between the TableItem, TableEditor and the customised widget, which causes problems when one is removed and the other related widgets need to be removed at the same time. Due to this added complexity, the TableItem is mapped to its TableEditor and customised widget.
The customised widget is named the MultiCheckSelectionCombo [29], due to its ability to select multiple values from a drop-down list of check-box styled options. It is similar to a Combo, but built from scratch because SWT conventions dictate that

the Combo class cannot be extended. The MultiCheckSelectionCombo is created with support for nearly all of the API of the Combo class, and both sets of Javadocs are available for comparison:

- Javadoc for MultiCheckSelectionCombo - link [28]
- Javadoc for Combo - link [12]

When the "Remove parameter" button is pressed:

1. Query the Table for selected parameters for a list of *selected* parameter TableItems
2. Retrieve corresponding TableEditor and SelectionCombo objects from mapping and remove them
3. Retrieve Parameter objects from mapping of TableItems to Parameter objects
4. Remove *selected* parameter TableItems from the Table for selected parameters
5. Create TableItem in the Table for available parameters for each Parameter object

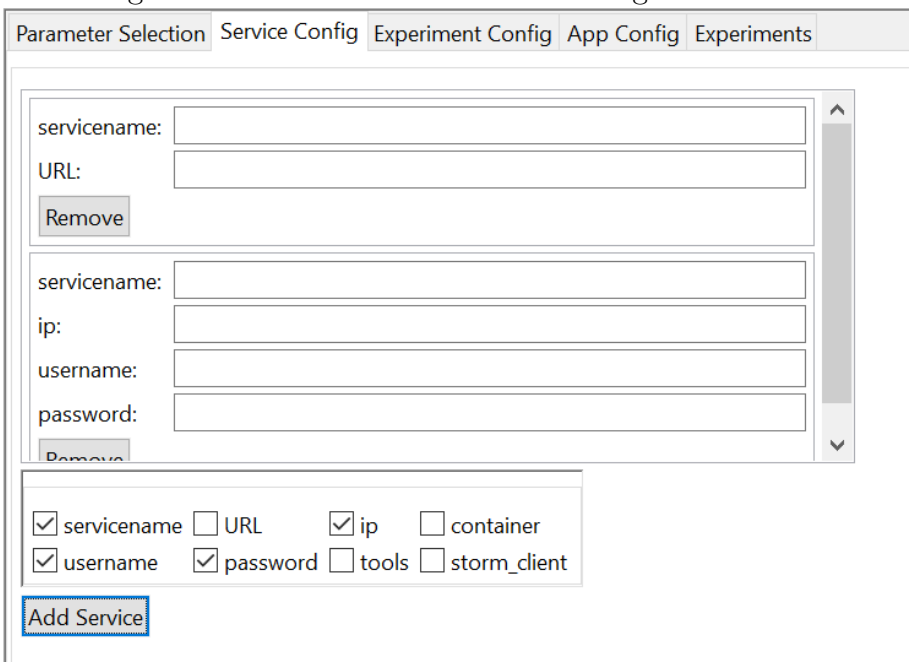
The changing of big data framework by selecting a different option on the drop-down list (Combo) is controlled by a *selection* listener. The listener's method is called whenever a new value is *selected*. The action removes all TableItems and related widgets (if any) from both Tables, with similar steps as mentioned above.

3.4.4 User Interface: Services and Configuration

The Services, Experiment, and Application configuration tabs are similar in that all of the user inputs are simple single text fields. The basic layout structure is a field name (SWT Label [13]) followed by a text box (SWT Text [13]) for the user to input. The Experiment and Application configuration tabs have no dynamic user interface content that needs to be generated in response to user interaction, and therefore are straight-forward to implement.
Although the Services Configuration tab similarly only accepts text input for each field, it requires dynamically generated user interface content, because a user can add an arbitrary number of services to connect to, and each service requires different fields to be specified.

As shown in Figure 12, the key innovation in the Service Configuration tab is

Figure 12: Screen shot of Service Configuration tab.



the use of a group of check boxes to customise each new Service. The Service class holds a SWT Text object for each of the 8 possible text fields. Due to the different number of fields that a service can hold, a Service Builder class manages the creation of Services and hides the constructor to prevent errors. When the "Add Service" button is pressed, the listener retrieves the selection status of the group of check boxes, and the iteratively builds a service using the Service Builder to create the required fields in the Service. The service is wrapped in a SWT Composite to manage its layout and provide a graphical representation of one coherent unit.

3.4.5 Configuration File Generation

The experiment configuration file (*expconfig.yaml*) is required input to start the Configuration Optimisation tool (*BO4CO*). The file has four sections corresponding to the details obtained from the user in the Eclipse plugin:

- Parameters - a selection of big data framework configruation parameters to experiment upon.
- Services configuration - details the connections to required services to deploy and monitor the experiment
- Experiment configuration - settings for experiment time limit, number of iteration and logging options
- Application configuration - details to enable execution of the big data application for experimentation

These information are available in the Eclipse plugin, which holds the set of Parameter objects and a set of user defined Service objects. The experiment and application configurations are not represented as objects, but have their values stored in the SWT Text object fields instead. These are read and collated into arrays of String arguments and passed to the file generator.
The file generator outputs these information with correct syntax according to the YAML specification. The overall structure of the file has four mappings for the four sets of details:

- A YAML Sequence of parameters, where each parameter is a set of simple YAML Scalars for each field.
- A YAML Sequence of services, where each service is a set of simple YAML Scalars for each field
- Two sets of simple YAML Scalars for the other experiemnt and application configurations

Upon testing with the BO4CO Configuration Optimisation tool, it was found that the tool had some flaws and incompleteness in its implementation:

- Hard coded reading of services information - The BO4CO code is hard coded to read the sequence of services in a fixed order, as listed here:
    1. Continuous Integration server
    2. DICE Deployment service
    3. DICE Monitoring service
    4. Storm cluster
    5. Kafka
    6. Zookeeper
    7. Hadoop cluster

In addition the the ordering problem, it also requires that all of the above services be present. That would be a rare case and seems to be a flaw, because a big data application would not typically use Storm and Hadoop frameworks at the same time.
The issue was resolved by fixing the ordering of services in the Eclipse plugin, and editing the BO4CO code to remove requirement of multiple big data frameworks.

- Numerical ranges not supported - The BO4CO configuration file appears to allow the specification of numerical ranges with an upper bound and lower bound. However, the actual code requires that each value for the experiment to be explicitly listed. For example, the range of *"1-5 with step 1"* would have to be listed as the discrete numerical options of {*1, 2, 3, 4, 5*}.
The issue was resolved by preprocessing numerical range parameters in the Eclipse plugin to generate such list of discrete numerical options.

3.5 Integration: Eclipse, Jenkins, BO4CO, Testbed

This section details the implementation to provide a fully integrated development environment that triggers configuration optimisation on remote automation server and runs tests on remote testbed. The following deliverables are covered:

- Integrate Eclipse and Jenkins for triggering parameterised builds with configuration files remotely
- Integrate Eclipse and Jenkins to retrieve and display BO4CO configuration results
- Integrate BO4CO tool and remote Storm testbed to deploy tests with different configuration parameters and retrieve performance metrics
- Integrate Jenkins automation server and MATLAB-based BO4CO tool to start experiment

3.5.1 Eclipse and Jenkins Integration

The aim is to provide integration between the Eclipse plugin described in the above sections, and a remote Jenkins server containing the Configuration optimisation tool instance. The tool requires an experiment configuration file to execute, and hence there is a need to remotely trigger Jenkins' parameterised build with a file parameter.
A Jenkins *project* has to be created, with remote triggering enabled. This opens the option to specify a token used for added security when remotely executing the build from the Jenkins API. The parameterised trigger option has to be selected, with a file parameter specified.
There is an existing Eclipse plugin that is capable of running builds and monitoring the status of remote Jenkins server, called the Hudson/Jenkins Mylyn Builds Connector [37]. Howver, this plugin does not support triggering parameterised builds and does not support sending file (or any) parameters. The possibility to re-use code from the plugin was investigated, but it was found that the code required to trigger parameterised build significantly differed from the existing code that only supported simple builds. Therefore the integration was build from scratch.
Jenkins offers an API that opens functionality to remote access. The API requires three security components to remotely trigger builds: [25]

- User-defined project remote trigger token
  The project token is a simple plain text token that is sent as one of the arguments in the HTTP request URL string.
- Credentials of an authorised user - username and password
  Pre-emptive authentication is used to authenticate the user with provided username and password. The pre-emptive authentication for Apache HTTP client is done by implementing a HTTPRequestInterceptor that intercepts all HTTP requests and injects the authentication component.

- CSRF
  There is no available implementation of the CSRF authentication for Jenkins and Apache HTTP client, possibly due to the fact that CSRF was only enforced recently for new installations of Jenkins 2.x upwards.
  The CSRF crumb can be obtained by requesting the Jenkins API's CrumbIssuer, and the responds message is parsed to extract the crumb value. It is stored and used in subsequent requests.

Remotely triggering parameterised builds is documented in the Jenkins remote access API. [25] Text based parameters such as Strings and integers were simple appended onto the trigger HTML request URL as parameters. However there was no documentation on how to include a file parameter [24], although there was a specific example using cURL [25]. Unfortunately the general requirement is unclear, and there are multiple ways of submitting a file as an HTML form using Apache HTTP client. Due to the lack of documentation, it was down to trial and error to find out which type of file submission did Jenkins support.
The file parameter is attached to the HTTP request as an HTTP entity created by a MultipartEntityBuilder, with the TextBody specify the plaintext file name in JSON, and file content attached as BinaryBody. As far as the trial and error results show, this exact approach is the only solution to triggering a Jenkins parameterised build remotely with a file parameter from a Java / Apache based HTTP request.
To retrieve the result from Jenkins Server to Eclipse plugin when the "Show results" button is pressed, an HTTP request is sent. Jenkins API provides a plaintext HTTP reponse that contains the entire console log output of the last successful build. The configuration optimisation results are extracted from this output.

3.5.2 Bug Fixes and Improvements to BO4CO

The Configuration optimisation tool (BO4CO) requires integration with the following services that it relies upon to function:

- DICE Jenkins Continuous Integration service - for monitoring experiments
- DICE Deployment Service - for deploying experiments
- DICE Monitoring Platform - to monitor performance of experiments
- Storm cluster - for executing experiments
- Zookeeper cluster - for deploying experiments
- Kafka cluster - for deploying experiments
- Hadoop cluster - for executing experiments

The integration with these services was designed to be included in the BO4CO code. However, upon testing the tool would not function properly. The reasons for malfunction is mainly due to incorrect initialisation, and outdated set up in connection with services, as detailed below:

- Hard coded reading of services information from experiment configuration file - The BO4CO code is hard coded to read the sequence of services in a fixed order in *init.m*, as listed here:

  1. Continuous Integration server
  2. DICE Deployment service
  3. DICE Monitoring service
  4. Storm cluster
  5. Kafka
  6. Zookeeper
  7. Hadoop cluster

  In addition the the ordering problem, it also requires that all of the above services be present. That would be a rare case and seems to be a flaw, because a big data application would not typically use Storm and Hadoop frameworks at the same time.
  The issue was resolved by fixing the ordering of services in the Eclipse plugin, and editing the BO4CO code to remove requirement of multiple big data frameworks.

- Incorrect access to non-existent field in HTTP response from the DICE Deployment service - The tool requests information from the DICE Deployment Service about the experimental Nimbus cluster, and attempts to access the *storm_nimbus_address.value* field from the HTTP response in *get_container_details.m*. It appears that the DICE Deployment Service's API has been updated, and the required information is now in the HTTP response field *storm_nimbus_host.value*.

- Erroneous calls to set up SSH connection - The tool attempts to set up SSH connections to services using the Ganymed SSH-2 library, in *setup.m*. The code calls *ssh2_setup()* to initialise the settings three times, which causes the first two connections settings to be overwritten and hence become unnecessary. The argument provided to the third call is also unsupported by the library. It causes the BO4CO code to run with a warning message that also appears in the currently released version of the tool. Upon investigation we were unable to infer the reason and possible thinking behind adding this argument to the method call. The solution is to remove the related code, so that the connection setup is only called once with the correct arguments (no arguments).

- Invalid selection of parameter options / Invalid array access - Following latin hypercube design, the tool attempts to run experiment on the parameter option at the generated index positions. However, the Latin hypercube design in *lhsdesign4grid.m* generates non-integer indices, causing the array access to be invalid. The solution is to round these indices to run experiment on the nearest parameter option.

3.5.3 Extension on BO4CO

The Big Data Auto-tuning tool aims to automatically improve big data application performance by configuration optimisation. The configuration parameters of big data frameworks fall into these four categories:

- Integer - the parameter may take any integer value between a set of lower and upper bounds.
- Percentage - the parameter may take any value between 0 and 1.
- Boolean - the parameter value may be TRUE or FALSE
- Categorical - the parameter may take any value from a list of Strings options

In the originally release version, BO4CO tool only supported numerical parameter values, i.e. Integer and Percentage parameters. In order to have "fully automatic" configuration optimisation, all parameters should be supported. Hence we extend the BO4CO tool to support Categorical and Boolean types.
The restriction on the existing implementation of BO4CO is an implementation restriction - an issue with data typing throughout the code. The main problem is not cause by the logical differences between numerical (continuous data) and categorical (discrete data). BO4CO treats numerical data as if it were discrete data, because it only considers the values provided in the *options* list. The code does not attempt to interpolate or extrapolate numerical values from the parameter options, nor does it attempt to apply any mathematical operation to the parameter options. In fact, one can argue that the core BO4CO algorithm is entirely oblivious to the value of the parameter option, and works strictly by selecting an index (that points to a parameter value), instead of directly selecting any values.
The parts of the tool which restricted data types of parameters include:

- Processing input - existing implementation parsed all parameter values into numerical vectors
- Initialisation - existing implementation used formatted strings requiring numerical values as arguments
- Connecting and deploying to Storm cluster - existing implementation of command string concatenation did not support data structures
- Data aggregation, logging and generating output - formatted output files required numerical arguments / arguments all of identical data type in a matrix

In order to maintain the format of output files for compatibility issues with other tools depending on the BO4CO configuration optimisation tool, the output stage required all parameter values to be numerical. This has been resolved by reporting the option's index number instead of its actual string value for categorical and boolean parameters. Other parts of the BO4CO code can be safely modified to support both data types, and uses the string value of the option when processing categorical and boolean parameters. The following steps were taken to implement this extension:

- Processing input - Input data parsed as vector of String using *strsplit* in *domain2option.m* instead of *str2num*.
- Initialisation - String generation using *strjoin* in *f_storm.m* instead of *fprintf* to remove formatting requirement
- Connecting and deploying to Storm cluster - Command generation using *strcat* and *char(cmd)* to create string, instead of primitive string concatenation using the *[ ]* operator.
- Data aggregation, logging and generating output - represent categorical and boolean options with their index value instead of string value, to maintain formatted output files that required numerical arguments / arguments all of identical data type in a matrix. If the string value is required, it can be obtained by checking the original experiment configuration file for list of options, and selecting the string value corresponding to the index.

3.5.4 Jenkins and BO4CO MATLAB Integration

Integrating the BO4CO tool to Jenkins is the final integration step that completes the fully automated Big Data Auto-tuning tool.
As mentioned in the above section, a Jenkins project is created with file parameter from the Eclipse plugin. By executing shell script or Windows batch commands on the Jenkins project, the file is moved into the working directory where the BO4CO code is located. The BO4CO tool can be run in deployed mode as a compiled MATLAB executable, and results are retrieved by access the output files at predefined location. To allow Eclipse plugin to retrieve the results, the file containing configuration optimisation results is printed on the console which is logged by the Jenkins API.
It is also possible to configure Jenkins to run with the BO4CO source code. This requires a MATLAB installation on the Jenkins server. To automate execution of BO4CO source code with Jenkins, the following arguments are required for the *matlab* command:

- *-nodisplay -nosplash -nodesktop* - To ensure that graphic elements are not displayed and a MATLAB editor is not started.
- *-r "main;exit"* - To run *main.m* and then exit MATLAB.
- *-wait* - To return only after the BO4CO experiments finish, instead of terminating when MATLAB has been successfully started.
- *-sd "source-code-directory"* - Provides path to source code directory to use as workspace.

Developers attempting to set up with Jenkins Servers on a Windows based system will have to take extra step. An unofficial guide exists for integrating MATLAB with Jenkins, which assumes the use of a Linux based system. There is no documentation on integrating MATLAB with a Windows based Jenkins installation. By investigation, it was found that the following two extra steps must be taken by developers that choose to use this set up:

- Jenkins must be installed as a standalone process. Jenkins for Windows is provided as an installer that installs as a system service. This creates an issue because the system processes cannot execute MATLAB. The solution is to run Jenkins from the command line with the user's access rights. Jenkins can be run as a standalone Java package, or wrapped in an off-the-shelf server instance such as Apache Tomcat.
- An extra argument is required to the command, that points to the MATLAB license file - *-c "path-to-license-file"*

4 Project Evaluation

4.1 Eclipse Plugin

4.1.1 UI Design choice

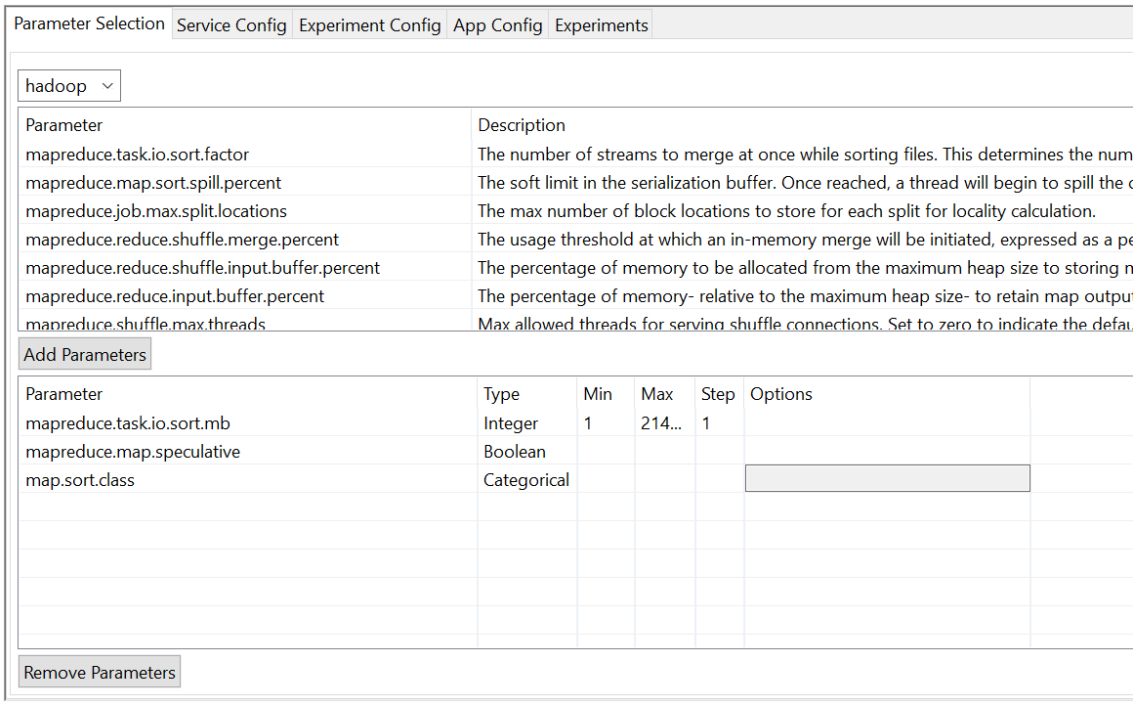
Engineering the user interface of the Eclipse plugin involved numerous design choices, and several iterations of the plugin UI as it evolved with feedback.
The selection of big data framework configuration parameters to experiment and optimise is the most important part of the Big Data Auto-tuning tool. It is also the most complex part of the user interface due to the types of parameters available, which leads to different details, different forms of display and selection required. The different layouts considered were:

1. A table for available parameters, and a list of customised composite widgets, one for each parameter selected. Customised composite widgets have different fields and forms of displays to suit different types of parameters.

2. Two tables, one showing available parameters, and one showing selected parameters. Options and details for each selected parameter can be viewed and edited by double-clicking to bring up a pop up settings box.

3. Two tables, one showing available parameters, and one displaying selected parameters and corresponding options and details. The currently implemented UI as shown in Figure 13.

Figure 13: Screen shot of parameter selection tab.



Each of the designs had their own merits and drawbacks:

1. Customised composite widgets are the most straight forward solution to handling parameters of different types. The ideal SWT widgets can be used to build up the composite for each parameter type. SWT has advanced built-in controllers to handle the layout and sizing of composites. However, the use of many individual widgets makes the layout crowded and bloated. Less parameters can be shown in the same space, and the different sizes and component for different parameters appears incoherent and messy to the user.

2. Having two tables gives the user interface a very coherent and clean layout. Tables are well suited to displaying strictly structured data. The incoherent details and options in the different types are hidden in the pop up setting boxes, that still allow each parameter type to be displayed using widgets that are an ideal fit. The drawbacks in this case is the inconvenience of having to double click and bring up a pop up box every time the user wishes to view or modify the details and options.

3. The current user interface layout is a balance between clean, coherency and convenience. The MultiCheckSelection Combo [29] was specifically designed to allow the options of categorical parameters to be semi-hidden with a drop-down menu, and fits the result into one cell. This allows the fitting of two tables that gives a clean, structured feel. There is a slight drawback in that forcefully attaching the created MultiCheckSelection Combo onto a table cell causes problems with the table's natural sizing, so some rows may not be automatically readjusted to their optimal size. It also is not entirely perfect in that the different types of parameters would leave blank columns used by other types of parameters. However, it was decided that these trade-offs are reasonable prices for a coherent layout that displays all information and details for each type of parameter concisely and conveniently. The table has added convenience of keyboard control and access using shortcuts such as the "TAB" and "ENTER" buttons to swiftly edit data.

Considerable feedback was received for the plugin's user interface, which helped evolve the parameter selection tab over three iterations.
The first suggestion was a filter of available parameters by its corresponding big data framework. This idea was swiftly adopted because it is convenient to the user in that they only see relevant parameters. It also improves the performance of the plugin, by not creating unnecessary entries, hence reducing the number of objects created and managed. The filter function also help ensure correctness of the parameter selection, preventing mixing up of parameters of different big data frameworks in the selection.
The second suggestion centered on how the user "selects" a parameter to be experimented upon. The original implementation uses a drag and drop gesture, where parameters can be dragged into the selected table, or dragged back to the available table. However, the performance of this gesture based system was problematic. It is believed that the custom combo used in one of the tables causes problems that may be due to the dynamic resizing that occurs whenever the table is changed. The feedback also suggests that a switch to having "Add Parameter" and "Remove parameter" buttons would be convenient for a broader group of developers especially

those who are using laptops without a proper mouse, or those who prefer to access UI components using the keyboard shortcuts. The drag and drop gesture does not work well for these user groups.
However, the additional buttons do create a problem, that they do need to be pressed when the users wishes to change selection. Previously, the user only had to perform one dragging jesture. To compensate, a double-click listener is added to the final version of the Eclipse plugin, so that the user has a more convenient way of selection.

4.1.2 UI Implementation Choices

Choice of framework and tools to work with is the most crucial when implementing the user interface. As the Big Data Auto-tuning tool is to be integrated with the Eclipse based DICE IDE environment, any work must be based on SWT. The approaches can be pure SWT or JFace on top of SWT:

- Eclipse SWT Standard Widget Toolkit [14]
  - Provides a common API across wide platform
  - Uses native widgets from the platform
  - Maintains native look and feel of user interface
  - Faster, more responsive and lighter system resource usage
  - Available as standard with Eclipse plugin development environment
  - Designed for Windows platform, other ports may be less efficient
  - Exposed to platform-specific bugs in the native library
  - Requires programmer to work with lower level native GUI library
  - Does not provide MVC architecture
- JFace [40]
  - brings Model-View-Controller architecture to SWT
  - provides adaptors to handle the common code for handling widget events
  - provides Actions to allow users to define their own behavior and to assign that behavior to specific components, e.g. menu items, tool items, push buttons, etc.
  - Provides registries that hold Images and Fonts
  - Defines standard dialogs and wizards, and defines a framework for building complex interactions with the user

The current implementation forgoes the majority of JFace uses, because the Eclipse plugin of the Big Data Auto-tuning tool functions mainly to obtain user input, and does not have to do too much processing on the input. The logic and data required in the plugin is not of the complexity to merit the use of the full MVC architecture, which although an elegant solution, would take additional time and effort to set up. The current implementation has a general architecture the separate the concerns of front end components responsible for displaying the user interface, and backend components to handle the integration functionalities. It is not envisioned that there will be significant extensions to the functionality of the Eclipse plugin that requires complex logic in the future. The Eclipse plugin also does not require any images or fonts, dialogs or wizards. Hence for the sake of simplicity, the only JFace components used in the current implementation are the adaptors to handle widget events.

Another architecture choice is on the use of the SWT's events and listener architecture. Under the listener architecture, each SWT component should be *listening*

to the events that relate to them. This is done by adding itself to the trigger's listener list. When an event occurs, it is broadcasted to all listeners in the list, by calling the corresponding method on the listener. It is up to each of the listener components to decide how it should react to the event by overriding the method.
The alternative implementation is for a central controller to react to events, and then execute actions on the components. Instead of the listeners subscribing to event broadcasts, the linking is reversed with the central controller holding information about all the relevant component it controls.
As detailed in Figure 11 in the implementation section, the current implementation is a combination of both approaches. There is not one central controller that coordinates all actions, but instead each key button on the tab controls its relevant actions. Buttons in the tab have listeners to detect when they are pressed, and then they control other components to execute the desired action. These other components include the tables, items in the tables and the customised selection combos attached to the tables. These components are stored in maps that are globally accessible throughout plugin context, so that controllers can access the relevant objects and data.
For example, to manage the removal of selected parameters:

1. The "Remove parameter" button listens for the event that it has been pressed
2. The listener method then queries the selected parameters table for a list of highlighted entries to be removed
3. Look up the maps to find the linked parameter and any linked graphical components such as customised selection combos.
4. Calls the disposal method on the linked components and the entry itself, and creates a new entry for the parameter in the available parameters table

This implementation is easy to understand as a programmer, and easy to debug and make modifications because all of the actions relating to this event of "removing a parameter" is centralised at that button.
An alternative implementation using the listener architecture:

1. Each entry in the selected parameters table subscribe to listen to the "Remove parameter" button
2. Each related graphical component to the entry listen to the entry
3. When the button is pressed all entries in the table are notified
4. Each entry then checks if it is highlighted to be removed
5. If the entry is highlighted, it in turn notifies its related graphical components
6. The graphical components will react by removing themselves

This implementation does not require extra memory space for global maps, but will create lists of listeners for each component. It also does not require looking up the

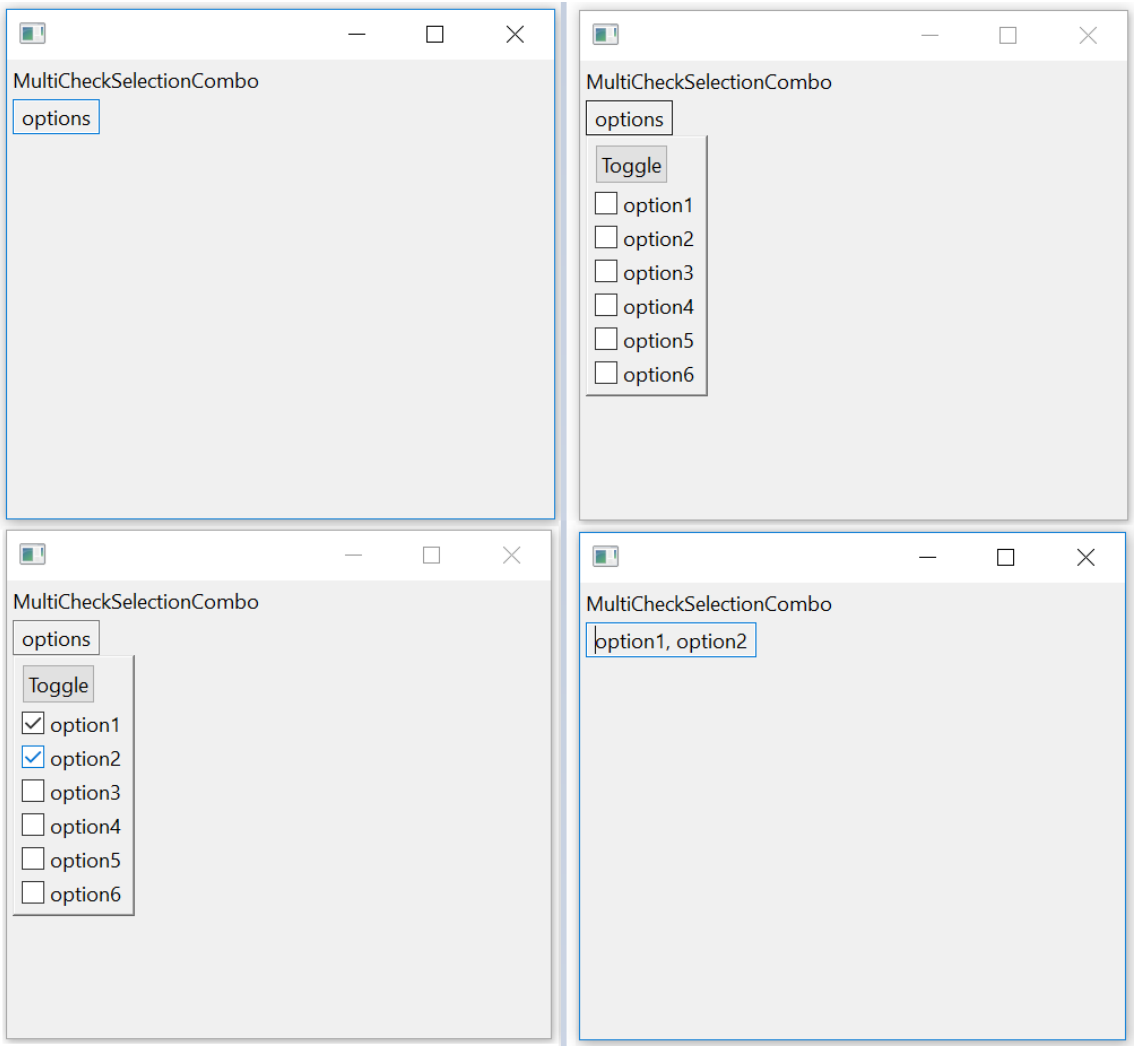
map to find corresponding components, but sends the notification to all components instead, leaving it to the component to decide if it needs to react. Due to the distributed nature of this approach, it is harder to debug what has gone wrong in the reaction to "removing a parameter". Developers have to update many components to make a single modification to the behaviour.
For these reasons, the choice was made to not fully implement the listener architecture down to the lower levels.

4.1.3 Contribution to SWT

The Eclipse Foundation's SWT Standard Widget Toolkit is open sourced and maintained by the community. In following of the open source spirit, the MultiSelectionCombo developed as part of this project has been released.
The MultiSelectionCombo was developed to allow selecting multiple values from a list of categorical options, required by to display and specify categorical parameters in the Big Data Auto-tuning tool. However it can be applied to any general use case where its features are required:

- shows an arbitrarily long list of options on a temporary drop down list
- each item on the list can be selected by selecting the corresponding check box button
- minimise to fixed size text display of selected options

Figure 14: Screen shot of MultiSelectionCombo.



The MultiSelectionCombo is similar to the SWT Combo, but it was built from scratch because SWT conventions dictate that the Combo class cannot be extended.

The Combo class does not allow multiple values to be selected, and does not contain the check box buttons required, hence there is a need for the creation of a new widget.
The case for this widget is further strengthened by the strong and persistent request for a combo that fits this specification - since as early as 2007 [26], there has been an activity in the community seeking for such a widget, and the requests have not stopped in the following years, up to 2015 [27].
There have been attempts and small, specific implementations of such a widget, but none with the clear objective of developing a general, re-usable SWT-standard widget [36]. The previous attemps also do not follow SWT conventions of standardised API, and fail to address efficiency, memory space and de-allocation considerations.
The MultiCheckSelectionCombo is created with support of all the API provided by the comparable original SWT Combo class. The MultiCheckSelectionCombo has clear documentation, simple and efficient data structure, and robust implementation of memory de-allocation to maintain performance. The detailed Javadoc of the MultiCheckSelectionCombo is available in the link [28], with the released source code available on GitHub [29].

4.1.4 Eclipse Plugin Backend

Big data framework configuration parameters are prone to change. Therefore it was an obvious decision to leave the specifics of configuration parameters out of the source code of the Big Data Auto-tuning Tool. The parameters are read from the *params.xml* file instead.
Currently one file contains all available parameters across all big data frameworks. It can be sensible to split into a different parameter file for each big data framework. That approach can lead to slightly faster loading time when showing parameters of selected big data framework, because the data is already filtered and does not require processing. However, for simplicity and convenience of implementation, it had been decided that the small performance gain was insignificant, given a list of parameters in the order of tens in total. Should the tool be extended to support more big data frameworks, or more configuration parameters become available, the option to split into multiple files should be re-considered.
It is also worth discussing the selection of parameters available in the Big Data Auto-tuning tool. The criteria for the inclusion of a configuration parameter was that it was "performance related". There has not been sufficient background research to clearly prove which parameters had significant impact on performance, and the selection was purely based on details provided by the official documentation. Given more time, the "performance relatedness" of a parameter could have been evaluated to take a more formulaic and scientific approach to selecting parameters to make available in the tool. An alternative would be to include all parameters and make it the developer's responsibility to select sensible options. However, that would be a backwards step from the aim to create and integrated development environment that aids developers with little or no knowledge to explore and improve the performance of big data applications.

4.2 Integration: Eclipse, Jenkins, BO4CO, Testbed

4.2.1 Eclipse and Jenkins Integration

Existing state of the art integration between Eclipse and Jenkins is provided by the Hudson/Jenkins Mylyn Builds Connector [37]. It does not support triggering of parameterised Jenkins builds from Eclipse, and hence also does not support the inclusion of file parameters in the trigger request.
In the wider scope, Jenkins remote API supports remote triggering of parameterised builds with file parameters [24], but there lack of documentation makes the task challenging.
The Big Data Auto-tuning tool is the first fully integrated tool that enables remote triggering of a Jenkins parameterised build with a file parameter from the Eclipse IDE. Although it is specifically designed to trigger the Configuration Optimisation process in its current form, it can be extended to support general use cases.

4.2.2 Bug Fixes and Improvements to BO4CO

A number of bugs were found in BO4CO code and fixed. Due to the complexity and tightly coupled nature of the system, with dependencies that extend beyond the code base into the services that are used to deploy experiments, parts of the debugging was time consuming. As the development was done in MATLAB, there were no advanced debugging and testing tools such as Java's JMock tester, which would have helped to eliminate problems and the very long running time of actually deploying real experiments on the Storm cluster to test. (At least 20 minutes per test).
Most of the fixes applied are ideal solutions which retain the intended functionality, are efficient and well designed. However, the fix to the erroneous hard-coded reading of service configurations in the initialisation step (*init.m*) should be regarded as a temporary fix, as a truly general solution has not been implemented due to time constraints.
There is a limitation on the validity of these fixes, as the testing has only been conducted on the source code run with a Storm cluster. There have been no tests run with a compiled version of the tool, and no tests with a Hadoop cluster, due to the lack of these resources. A compiled version was unavailable due to MATLAB licensing issue.

4.2.3 Extension on BO4CO

BO4CO Configuration optimisation tool is a ground breaking approach to help developers of big data applications automatically improve the performance by experimenting on configuration parameters. It was limited to experimenting with numerical parameters. This work to extend its functionality to categorical and boolean parameters is yet another ground breaking step, such that it can now be said that the BO4CO Configuration optimisation tool can work with all existing big data framework parameters. The investigation concluded that all parameters for all big data frameworks considered in the DICE project are of one of the four supported types: Integer, Percentage, Categorical or Boolean. Hence this removes a significant restriction on the vision of a fully integrated tool that automates the entire process of configuration optimisation.
It is believed that the algorithm has no problems dealing with categorical and boolean data types, despite the fundamental differences with numerical data types. The algorithm processes index values of the options provided, instead of dealing directly with the value of the option itself [31]. Despite the fact the options for numerical parameters may show a sense of pattern (e.g. an increasing order), while categorical and boolean parameters are perceived to have no such ordering, the algorithm does not rely on such ordering to function, so should function correctly for these new data types. It is accepted that the ordering of numerical options does helped the performance of the tool, but in the long run the results of non-ordered options are believed to converge as well.
Due to time constraints, it was not possible to draw any meaningful statistical measure of how well the tool performs with Categorical and Boolean parameters. However, the implementation has been validated - the tool is able to process and deploy experiments with Categorical and Boolean parameters, with no adverse effect to the existing functionality that supported numerical parameters.

4.2.4 Jenkins and BO4CO MATLAB Integration

Jenkins provides the ability to automatically execute the tool in a remote server, monitor the progress and retrieve the results any time after the experiments have completed. The remote, automatic and self-contained nature of a Jenkins server is crucial to the Big Data Auto-tuning Tool, as the configuration optimisation experiments can run for any time from tens of minutes up to tens of hours. It would be a significant inconvenience if the developer had to run the process on the local machine, and possibly keep the Eclipse plugin open for the whole duration.
The work on this integration was specific for the use case, but it provides a consolidated approach that can be generalised to all developers who wish to set up Jenkins automation and continuous integration for running and testing MATLAB source code in a Windows environment. There has not been a comprehensive and detailed solution for this setting, with the community documentation [2] only introducing the approach for Linux environments, that does not directly work in a Windows set up.

5 Conclusion

5.1 Conclusion

The Big Data Auto-Tuning tool has been implemented as a fully integrated solution to finding optimum configuration for big data applications. It consists of a neat user interface in the Eclipse IDE, integrated to a Jenkins server running the BO4CO configuration optimisation tool, that deploys and monitors experiments on a Storm cluster, and retrieves results to the developer on the Eclipse plugin UI.
In doing so, the project has overcome challenges to achieve:

- - A development environment in the Eclipse IDE, where developers can explore configuration optimisation and the performance of their applications.
  - A new SWT-based widget, MultiSelectionCombo, that conforms to conventions and standards, and has been published to the open source community
  - Extension to the BO4CO code to further this state of the art configuration optimisation tool to support Categoric and Boolean parameters in addition to numerical parameters
- - Fully integrated development environment that triggers configuration optimisation on remote automation server and run tests on remote testbed.
  - Integration between Eclipse and Jenkins for triggering parameterised builds with configuration files remotely, providing an example that integration between the two can be deeper and cover more advanced functionality than existing solutions.
  - Integrated Jenkins automation server and MATLAB-based BO4CO tool, demonstrating a new viable continuous integration path for MATLAB and Jenkins on Windows-based systems.

5.2 Future Works

Future works relating to this project can explore the following directions:

- Big data framework configuration parameters
  The criteria for inclusion of configuration parameters for this project can be improved with a better understanding of the underlying big data frameworks, and how the parameters are designed to affect operations. There can be more experimentation to collect statistics and more scientifically determine whether a configuration parameter would affect performance of big data applications. This data can also be used to better present the available parameters, by ordering and sorting them for different use cases, providing helpful hints to developers with less knowledge, and provide realistic limits and guideline figures to aid selection.

- Extra features on the Eclipse plugin

  - Improved visualisation and analysis of results - Configuration optimisation results can be accompanied with statistics to show how performance has improved over experiment runs.
  - Saved session and settings - Frequently used settings can be saved and loaded for convenience. The services and test settings usually stays the same across many configuration optimisation experiment runs, only with the parameters changed. The developer will not need to always re-enter such information.

- Further work on BO4CO configuration optimisation tool

  - Complete testing for Hadoop big data framework - Fixes to the tool have not been tested on Hadoop
  - Investigate the performance of configuration optimisation with categorical and boolean parameters - The new capability has been implemented but lacks sufficient testing to statistically validate its performance. There is every reason to believe that it will perform similarly to how it has done with numerical parameters, possibly with a less predictable performance at the initial iterations, but converging to optimal solution after the initial stage.

5.3 Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr Giuliano Casale. His guidance to shape the direction of the project, unwavering assistance in fixing the BO4CO tool, and deep knowledge in the project area was invaluable in making this project a success.
I would also like to thank Matej Artač for his kind assistance in setting up the numerous tools required; and my second marker, Dr Robert Chatley, for his helpful advice and suggesting new perspectives from which to evaluate the project.

6 Appendix

6.0.1 Parameters XML schema

Figure 15: Parameters XML schema.

```
<xs:element name="params">
   <xs:complexType>
     <xs:choice maxOccurs="unbounded" minOccurs="0">
       <xs:element name="intparam" maxOccurs="unbounded" minOccurs="0">
         <xs:complexType>
           <xs:sequence>
             <xs:element type="xs:string" name="name"/>
             <xs:element type="xs:string" name="node"/>
             <xs:element type="xs:int" name="default"/>
             <xs:element type="xs:int" name="lowerbound"/>
             <xs:element type="xs:int" name="upperbound"/>
             <xs:element type="xs:string" name="description"/>
           </xs:sequence>
         </xs:complexType>
       </xs:element>
       <xs:element name="boolparam" maxOccurs="unbounded" minOccurs="0">
         <xs:complexType>
           <xs:sequence>
             <xs:element type="xs:string" name="name"/>
             <xs:element type="xs:string" name="node"/>
             <xs:element type="xs:string" name="default"/>
             <xs:element type="xs:string" name="description"/>
           </xs:sequence>
         </xs:complexType>
       </xs:element>
       <xs:element name="catparam" maxOccurs="unbounded" minOccurs="0">
         <xs:complexType>
           <xs:sequence>
             <xs:element type="xs:string" name="name"/>
             <xs:element type="xs:string" name="node"/>
             <xs:element type="xs:string" name="default"/>
             <xs:element type="xs:string" name="option" maxOccurs="unbounded" minOccurs="0"/>
             <xs:element type="xs:string" name="description"/>
           </xs:sequence>
         </xs:complexType>
       </xs:element>
     </xs:choice>
   </xs:complexType>
 </xs:element>
```

6.0.2 Controls in SWT

[6]

Widget	Purpose
Browser	Control containing a native HTML renderer.
Button	Selectable control that issues notification when pressed andor released.
Canvas	Composite control that provides a surface for drawing arbitrary graphics. Often used to implement custom controls.
Caret	An i-beam that is typically used as the insertion point for text.
Combo	Selectable control that allows the user to choose a string from a list of strings, or optionally type a new value into an editable text field.
Composite	Control that is capable of containing other widgets.
CoolBar	Composite control that allows users to dynamically reposition the cool items contained in the bar.
CoolItem	Selectable user interface object that represents a dynamically positionable area of a cool bar.
DateTime	Selectable user interface object that allows the user to enter and modify date or time values.
ExpandBar	Composite control that groups pages that can be shown or hidden by the user with labeled headers.
ExpandItem	Selectable user interface object corresponding to a header for a page in an ExpandBar.
Group	Composite control that groups other widgets and surrounds them with an etched border andor label.
Label	Non-selectable control that displays a string or an image.
Link	Selectable control that displays a text with links.
List	Selectable control that allows the user to choose a string or strings from a list of strings.
Menu	User interface object that contains menu items.
MenuItem	Selectable user interface object that represents an item in a menu.
ProgressBar	Non-selectable control that displays progress to the user, typically in the form of a bar graph.
Sash	Selectable control that allows the user to drag a rubber banded outline of the sash within the parent window. Used to allow users to resize child widgets by repositioning their dividing line.
Scale	Selectable control that represents a range of numeric values.
ScrollBar	Selectable control that represents a range of positive numeric values. Used in a Composite that has V_SCROLL andor H_SCROLL styles.
Shell	Window that is managed by the OS window manager. Shells can be parented by a Display (top level shells) or by another shell (secondary shells).

Widget	Purpose
Slider	Selectable control that represents a range of numeric values. A slider is distinguished from a scale by providing a draggable thumb that can adjust the current value along the range.
Spinner	Selectable control that allows the user to enter and modify numeric values.
TabFolder	Composite control that groups pages that can be selected by the user using labeled tabs.
TabItem	Selectable user interface object corresponding to a tab for a page in a tab folder.
Table	Selectable control that displays a list of table items that can be selected by the user. Items are presented in rows that display multiple columns representing different aspects of the items.
TableColumn	Selectable user interface object that represents a column in a table.
TableItem	Selectable user interface object that represents an item in a table.
Text	Editable control that allows the user to type text into it.
ToolBar	Composite control that supports the layout of selectable tool bar items.
ToolItem	Selectable user interface object that represents an item in a tool bar.
Tree	Selectable control that displays a hierarchical list of tree items that can be selected by the user.
TreeColumn	Selectable user interface object that represents a column in a tree.
TreeItem	Selectable user interface object that represents a hierarchy of tree items in a tree.

6.0.3 Events in SWT

[10]

Low level events

Event Type	Description
FocusIn, FocusOut	A control has gained or lost focus.
KeyDown, KeyUp	The user has pressed or released a keyboard key when the control has keyboard focus.
MouseDown, MouseUp, MouseDoubleClick	The user has pressed, released, or double clicked the mouse over the control.
MouseMove	The user has moved the mouse above the control.
MouseEnter, MouseExit, MouseHover	The mouse has entered, exited, or hovered over the control.
MouseHorizontalWheel, MouseVerticalWheel, MouseWheel	The mouse wheel has been rotated.
Paint	The control has been damaged and requires repainting.
Touch	The user has touched a touch-based input source over the control.

High level events

Event Type	Description
Activate, Deactivate	Generated when a Control is activated or deactivated.
Arm	A MenuItem is armed (highlighted and ready to be selected).
Close	A Shell is about to close as requested by the window manager.
DefaultSelection	The user selects an item by invoking a default selection action. For example, by hitting Enter or double clicking on a row in a Table.
Dispose	A widget is about to be disposed, either programmatically or by user.
DragDetect	The user has initiated a possible drag operation.
EraseItem	A TableItem or TreeItem is about to have its background drawn.
Expand, Collapse	An item in a Tree is expanded or collapsed.
Gesture	The user has used a touch-based input source to perform a gesture over the control.
Help	The user has requested help for a widget. For example, this occurs when the F1 key is pressed under Windows.
Iconify, Deiconify	A Shell has been minimized, maximized, or restored.
ImeComposition	Allows custom text editors to implement in-line editing of international text.
MeasureItem	The size of a custom drawn TableItem or TreeItem is being requested.
MenuDetect	The user has requested a context menu.
Modify	The widget's text has been modified.
Move, Resize	A control has changed position or has been resized, either programmatically or by user.
Movement	An updated caret offset is needed in response to a user action in a StyledText.
OpenDocument	The operating system has requested that a document be opened.
OrientationChange	The orientation of a Text control is changing.
PaintItem	A TableItem or TreeItem is about to have its foreground drawn.
Selection	The user selects an item in the control. For example, by single clicking on a row in a Table or by keyboard navigating through the items.
SetData	Data needs to be set on a TableItem when using a virtual table.
Settings	An operating system property, such as a system font or color, has been changed.
Show, Hide	A control's visibility has changed.
Skin	A control needs to be skinned.
Traverse	The user is trying to traverse out of the control using a keystroke. For example, the escape or tab keys are used for traversal.
Verify	A widget's text is about to be modified. This event gives the application a chance to alter the text or prevent the modification.

References

[1] BTO Research Technical Briefing. *'DevOps@Unicredit*. 2015.

[2] Andy Campbell. *The Other Kind of Continuous Integration*. URL: http://blogs.mathworks.com/developer/2015/01/20/the-other-kind-of-continuous-integration/ (visited on 19/06/2017).

[3] Giuliano Casale. *expconfig.yaml*. URL: https://github.com/dice-project/DICE-Configuration-BO4CO/blob/master/src/conf/expconfig.yaml (visited on 19/06/2017).

[4] DICE. *DICE Project*. 2016. URL: http://www.dice-h2020.eu/ (visited on 10/02/2017).

[5] DICE. *DICE Project Tools*. 2016. URL: http://www.dice-h2020.eu/tools (visited on 10/02/2017).

[6] Eclipse. *Controls*. URL: http://help.eclipse.org/neon/topic/org.eclipse.platform.doc.isv/guide/swt_widgets_controls.htm?cp=2_0_7_0_0 (visited on 19/06/2017).

[7] Eclipse. *Eclipse IDE*. URL: http://www.eclipse.org/ide/ (visited on 19/06/2017).

[8] Eclipse. *Eclipse Marketplace*. URL: https://marketplace.eclipse.org/ (visited on 19/06/2017).

[9] Eclipse. *Eclipse Packages*. URL: https://www.eclipse.org/downloads/eclipse-packages/ (visited on 19/06/2017).

[10] Eclipse. *Events*. URL: http://help.eclipse.org/neon/topic/org.eclipse.platform.doc.isv/guide/swt_widgets_events.htm?cp=2_0_7_0_1 (visited on 19/06/2017).

[11] Eclipse. *Layouts*. URL: http://help.eclipse.org/neon/topic/org.eclipse.platform.doc.isv/guide/swt_layouts.htm?cp=2_0_7_1 (visited on 19/06/2017).

[12] Eclipse. *org.eclipse.swt.widgets: Class Combo*. URL: http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fswt%2Fwidgets%2FCombo.html (visited on 19/06/2017).

[13] Eclipse. *SWT Widgets*. URL: http://www.eclipse.org/swt/widgets/ (visited on 19/06/2017).

[14] Eclipse. *The Standard Widget Toolkit*. URL: http://help.eclipse.org/neon/topic/org.eclipse.platform.doc.isv/guide/swt.htm?cp=2_0_7 (visited on 19/06/2017).

[15] Eclipse. *Widgets*. URL: http://help.eclipse.org/neon/topic/org.eclipse.platform.doc.isv/guide/swt_widgets.htm?cp=2_0_7_0 (visited on 19/06/2017).

[16] Gartner. *'Research Note'*.

[17] Tatiana Ustinova Giuliano Casale. *DICE: State of the Art Analysis*. 2015. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2015/08/D1.1_State-of-the-art-analysis1.pdf (visited on 19/06/2017).

[18] vogella GmbH. *Eclipse IDE Plug-in Development: Plug-ins, Features, Update Sites and IDE Extensions*. URL: http://www.vogella.com/tutorials/EclipsePlugin/article.html (visited on 19/06/2017).

[19] Apache Hadoop. *core-default.xml*. URL: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/core-default.xml (visited on 19/06/2017).

[20] Michael Hüttermann. *DevOps for Developers'. Apress*. 2012.

[21] jaks. *Eclipse says 'Class not found' even though the classes are available*. URL: https://stackoverflow.com/questions/11077292/eclipse-says-class-not-found-even-though-the-classes-are-available (visited on 19/06/2017).

[22] Jenkins. *Plugins Index*. URL: https://plugins.jenkins.io/ (visited on 19/06/2017).

[23] Kohsuke Kawaguchi. *Hudson*. URL: http://kohsuke.org/ (visited on 19/06/2017).

[24] Kohsuke Kawaguchi. *Parameterized Build*. URL: https://wiki.jenkins-ci.org/display/JENKINS/Parameterized+Build (visited on 19/06/2017).

[25] Kohsuke Kawaguchi. *Remote access API*. URL: https://wiki.jenkins-ci.org/display/JENKINS/Remote+access+API (visited on 19/06/2017).

[26] Kevin. *Multiple select combo?* URL: http://www.eclipsezone.com/eclipse/forums/t100279.html (visited on 19/06/2017).

[27] Suhail Ahmed Khan. *How to add a checkbox in a combo(Drop down) in java swt?* URL: https://stackoverflow.com/questions/31673121/how-to-add-a-checkbox-in-a-combodrop-down-in-java-swt (visited on 19/06/2017).

[28] Derek Law. *Class MultiCheckSelectionCombo*. URL: https://lawhcd.github.io/SWTMultiCheckSelectionCombo/ (visited on 19/06/2017).

[29] Derek Law. *SWTMultiCheckSelectionCombo*. URL: https://github.com/lawhcd/SWTMultiCheckSelectionCombo (visited on 19/06/2017).

[30] OSGi. *OSGi Certified Products*. URL: https://www.osgi.org/osgi-compliance/osgi-certification/osgi-certified-products/ (visited on 19/06/2017).

[31] Giuliano Casale Pooyan Jamshidi. *An Uncertainty-Aware Approach to Optimal Configuration of Stream Processing Systems*. 2015. URL: https://arxiv.org/pdf/1606.06543v1.pdf (visited on 10/02/2017).

[32] Matej Artač Pooyan Jamshidi Giuliano Casale. *BO4CO: Bayesian Optimization for Configuration Optimization*. 2017. URL: https://github.com/dice-project/DICE-Configuration-BO4CO (visited on 19/06/2017).

[33] Maciej Bukowski Sonia Buchholtz. *Big and open data in Europe - A growth engine or a missed opportunity?* 2014. URL: demosEUROPA.

[34] Apache Storm. *org.apache.storm: Class Config*. URL: http://storm.apache.org/releases/2.0.0-SNAPSHOT/javadocs/org/apache/storm/Config.html (visited on 19/06/2017).

[35] Apache Storm. *Tutorial*. URL: http://storm.apache.org/releases/current/Tutorial.html (visited on 19/06/2017).

[36] suxiaogang. *MultipleSelectionCombo.java*. URL: https://gist.github.com/suxiaogang/6311176d2c5b9a8d1867 (visited on 19/06/2017).

[37] Paul Verest. *Hudson/Jenkins Mylyn Builds Connector*. URL: https://marketplace.eclipse.org/content/hudsonjenkins-mylyn-builds-connector (visited on 19/06/2017).

[38] David Reinsel Vernon Turner John F. Gantz. *The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things*. URL: Report%20from%20IDC%20for%20EMC%20April%202014.

[39] Matt Weinberger. *DevOps explained: A philosophy of speed, not momentum.*

[40] Wikipedia. *JFace*. URL: https://en.wikipedia.org/wiki/JFace (visited on 19/06/2017).