Dynamically Reconfigurable Workflows for Time-Critical Applications

Kieran Evans, Andrew Jones, Alun Preece, Francisco Quevedo, David Rogers, Irena Spasić, Ian Taylor
School of Computer Science and Informatics
Cardiff University, UK
+44 2920 874812
TaylorIJ@cardiff.ac.uk

Vlado Stankovski, Salman Taherizadeh, Jernej Trnkoczy
Faculty of Civil and Geodetic Engineering
University of Ljubljana, Slovenia
+386 1 4768 588
Vlado.Stankovski@fgg.uni-lj.si

George Suciu, Victor Suciu
BEIA Consult International, Romania
+40213323006
george@beia.ro

Paul Martin, Junchao Wang, Zhiming Zhao
Informatics Institute
University of Amsterdam, the Netherlands
+31 20 5257599
z.zhao@uva.nl

ABSTRACT

Cloud-based applications that depend on time-critical data processing or network throughput require the capability of reconfiguring their infrastructure on demand as and when conditions change. Although the ability to apply quality of service constraints on the current Cloud offering is limited, there are on-going efforts to change this. One such effort is the European funded SWITCH project that aims to provide a programming model and toolkit to help programmers specify quality of service and quality of experience metrics of their distributed application and to provide the means to specify the reconfiguration actions which can be taken to maintain these requirements. In this paper, we present an approach to application reconfiguration by applying a workflow methodology to implement a prototype involving multiple reconfiguration scenarios of a distributed real-time social media analysis application, called Sentinel. We show that by using a lightweight RPC-based workflow approach, we can monitor a live application in real time and spawn dependency-based workflows to reconfigure the underlying Docker containers that implement the distributed components of the application. We propose to use this prototype as the basis for part of the SWITCH workbench, which will support more advanced programmable infrastructures.

Categories and Subject Descriptors

B.8.1 [**Performance**]: B.8.1 Reliability, Testing, and Fault-Tolerance.

Keywords

Workflows, Dynamic Data Driven Systems; Quality of Service; Quality of Experience; Time-Critical Applications.

SC'15, Austin, Texas, USA.

DOI: http://dx.doi.org/10.1145/2822332.2822339

1. INTRODUCTION

Computation has revolutionized the way science and engineering are conducted by fast becoming their third branch alongside theory and experiment. Cloud computing and supercomputing facilities have enabled extremely complex business and scientific data analysis, visualization pipelines and time critical applications to be developed. Time critical applications generally require rapid reconfiguration in response to time-sensitive events over a prolonged period. Within Cloud environments both the development and operation of time critical applications are very difficult because Clouds generally do not offer explicit Quality of Service (QoS) for networking capabilities and thus it is impossible for systems to categorically maintain a guaranteed level performance or service quality. Indeed, even if the bandwidth or CPU processing power is far higher than what is required by an individual application, the aggregate of all concurrently run applications may surpass the inherent limitations.

The distributed nature and QoS requirements, along with their sophisticated optimization mechanisms, make time-critical applications particularly difficult to support. Although cloud environments are capable of providing virtualized, elastic, controllable and high quality on-demand services, they still lack the ability to control the selection and configuration of infrastructural components in response to changing requirements and environmental pressures. Specifically, current Cloud environments still do not have the tools and application programming interfaces that would allow the developers to exert such control on the underlying infrastructure in an intelligent, semi-autonomous manner.

Workflow systems have increasingly been employed in scientific research to automate the research lifecycle which often involves a repetitive set of tasks including; code development, data refinement, migration to an appropriate computing infrastructure, execution, and retrieval and analysis of the results. Workflows provide a structured means of describing complex functional execution and data pipelines for a scientific experiment. By modelling the specification of the scientific process using sub-elements of a task, each of which can be independently developed, validated, refined and composed for different

configurations [1], workflows expose the underlying scientific processes, thereby enabling the reproducibility of results.

In this paper we describe the marriage of a Web-based workflow orchestration and execution environment, called WZeroRPC [2] to a time-critical Social Media analysis application, called Sentinel [3], in order to create a prototype reconfigurable application and environment that adapts to the ever changing processing requirements encountered as discussion on topics of interest rise and fall on Social Media. WZeroRPC allows multiple distributed workflows to be connected together, and the logic of how one coordinates the workflows is specified through a simple Python API. Sentinel provides a dynamic and scalable approach to extract near real time meaning from Big Data, allowing existing analytic tools to analyse social media data in a real time fashion.

For this use case, we provide a workflow implementation that can programmatically monitor metrics from the live running Sentinel application and reconfigure the application for two specific scenarios: the first, where a new social media search is added to the processing pipeline; and the second, where a search suddenly increases in size. Both scenarios require reconfiguration of several components. The application is distributed using Docker[1] to provide complete controllability of each service.

Using this approach, our solution provides two novel contributions: the first contribution describes a mechanism which is capable of reconfiguring the Dockerised infrastructure on demand. The second contribution describes a novel workflow approach, which is capable of monitoring the live application and instructing the reconfiguration subsystem to restructure itself upon reaching certain conditions.

We also discuss how this kind of approach can be built upon via the objectives of the SWITCH project. The Software Workbench for Interactive, Time Critical and Highly self-adaptive Cloud applications is a project that aims to go much further than the prototype presented here. It attempts to address the issues of maintaining QoE and QoS by providing a new software methodology that can improve the development and execution of time-critical applications in Clouds. We argue that the WZeroRPC-Sentinel prototype can provide a basis for the initial prototype implementation for SWITCH, by addressing the re-configurability requirements for this platform through the control of multiple co-existing workflow instances via a centralised engine capable of coordinating the logic of execution by monitoring metrics and dynamically instantiating workflows to manage the reconfiguration requirements of the underlying infrastructure.

The rest of the paper is organized as follows: Section 2 describes the state of the art in the area of time-critical applications and Workflow Management Systems (WMS). Section 3 provides an overview of the SWITCH project in the context of which this work has been done. Section 4 outlines the benefits of the WZeroRPC workflow system and compares it to other WMS in order to implement the provided scenario. Section 5 gives an overview of the Sentinel application scenario case and its implementation. Section 6 briefly presents further use cases in the SWITCH project and finally Section 7 draws a conclusion for this work.

2. STATE OF THE ART

Programming distributed applications often depends on the adoption of a specific computing architecture or platform; typical examples include Message Passing Interface (MPI)-based parallel computing in distributed memory cluster architecture, service platform-based workflow applications, and Cloud-based Map Reduce processing. Time critical applications may likewise involve some MPI or other parallel computing-based components for high performance data processing; however, the distributed nature of the system components often makes the time critical application difficult to deploy in a parallel computing program based on technologies like MPI. Quality constraints are used in workflow applications for describing the abstract workflows, and for creating the runtime enactment, such as in [4]. However, in those applications, the creation of the application logic is mostly separated from the customisation of the runtime environment; in particular, a formal model is rarely utilised in verifying time constraints.

At the same time, Workflow Management Systems (WMS) have been used for modelling and executing rigidly structured scientific or business processes (workflows) in local or distributed environments. However, the reconfiguration at runtime of the applications and/or the infrastructure based on the results of certain decision-making procedures has rarely been taken into account by current workflow management systems.

Below we present a list of the most well-known current WMS and some of their features:

Taverna[2] is an open source and domain-independent WMS for designing and executing scientific workflows and aiding *in silico* experimentation. Taverna was created by the myGrid team and been accepted at the end of 2014 as an Apache incubator project.

Taverna automates experimental methods through the use of a number of different (local or remote) services from a diverse set of domains [5]. A Taverna workflow can invoke general SOAP/WSDL or REST Web services, run R scripts in local or remote Rserve, run Java code as beanshell, and invoke external tools on local and remote machines (via SSH).

Triana[3] is an open source problem solving environment developed at Cardiff University that combines an intuitive visual interface with powerful data analysis tools. Unlike many workflow systems, Triana is decentralized and distributes several control units over different computing resources through a modularized architecture consisting of a cooperating collection of interacting components. Under a typical usage scenario, clients may log into a Triana Controlling Service (TCS), remotely compose and run a Triana application and then visualize the result locally – even though the visualization unit itself is run remotely [6].

Pegasus[4] helps workflow-based applications to execute in a number of different environments including desktops, campus clusters, grids, and clouds [7]. Created at the University of Southern California and licensed under the Apache 2 licence, Pegasus provides APIs in Java, Python and Perl that allow users to develop abstract DAG based workflows in XML, known as DAX files.

[1] https://www.docker.com/

[2] http://www.taverna.org.uk/

[3] http://www.trianacode.org/

[4] http://pegasus.isi.edu/

Pegasus provides a means of workflow resilience, by retrying elements of the workflow upon failure. This can be at task level or can involve retrying the entire workflow. Workflow-level checkpointing, re-mapping of portions of the workflow, and attempting to run alternative data sources for staging data, are all other means by which Pegasus attempts to solve workflow issues. If unsuccessful, Pegasus will provide a rescue workflow containing a description of only the work that remains to be done.

ASKALON[5], developed by the University of Innsbruck, is a Grid and Cloud environment for composition and execution of scientific workflow applications. The primary optimisation goal of the ASKALON environment is the reduction of workflow execution time [8].

The ASKALON environment consists of a Workflow composition interface, a collection of runtime middleware services and tools that allow the running of scientific workflows on the Austrian Grid and both the Eucalyptus and Amazon EC2 cloud infrastructures.

MOTEUR, developed by CNRS, has adopted the Simple Concept Unified Flow Language (Scufl) used by Taverna 1x as the workflow composition language. The designers highlight the strength that Scufl has in describing data flows used within the e-Science community and the concept of coordination constraints; control links that enforce order of execution between non-dependant services enabling the identification of data synchronisation requirements [9].

Workflows created in MOTEUR can be serialised to XML files based upon the GWENDIA language; a workflow language allowing a coherent integration of a data-driven approach, arrays manipulation and control structures. A GWENDIA workflow is a graph of nodes interconnected through dependency links. These nodes are atomic units, each representing an application service that is bound to an arbitrary number of input and output ports. Nodes can be: (i) a web service wrapped by GASW (gLite, OAR or local execution) or jGASW (gLite and local execution), (ii) a DIET service invoked through DIET API, or (iii) a piece of local Java code (beanshell).

None of the WMS mentioned above allow the composition of dynamic data-driven scenarios, where one workflow may process data until a certain condition occurs, and then another workflow take the output data of the first workflow, process it and adapt the application accordingly to certain criteria, taking advantage of the programmability of the infrastructure, in order to ensure that the QoS of the application is maintained. However a workflow system that can do these things has the potential to be extremely useful in elastic virtualised environments, especially for time-critical applications. In Section 4 we shall present WZeroRPC, which addresses this issue, and also explain its relationship to other WMS more generally. However, we shall first explain the context within which this work has been carried out, namely the SWITCH project.

3. SWITCH

Advanced infrastructures enable quality guaranteed runtime environments for time critical applications, in which we can see two important foci directly related to time critical applications. The first one is from the transferral of High Performance Computing (HPC) environments to virtualized infrastructure such as the Cloud, for instance HPC cloud services in the European Grid Initiative (EGI)[6]. In those environments, supercomputers, and HPC and GPU clusters, are deployed in a cloud environment to support tasks with very high performance requirements, but underlying most of them is an MPI-based parallel computing model. The second focus is driven by the emergence of advanced network technologies—not only advanced hardware (e.g. quality-guaranteed optic networks) but also advanced protocols for controlling network behaviour and quality of service, such as Software Defined Networking (SDN) [10]. These advanced infrastructures provide developers opportunities to program and customise qualified runtime environments for time critical applications. However, to do this effectively also requires 1) an effective planning model for defining the virtual runtime environment, 2) advanced network services for optimised communication, and 3) technologies for issuing Service Level Agreements (SLAs) and real-time negotiation.

To take advantage of the next generation of elastic, virtualized infrastructure, we need to take the architecture described in this paper and extend it into a fully featured workbench supporting co-programming of both application and infrastructure.

In detail, SWITCH aims to provide:

1) **An effective planning model for defining virtual runtime environment** involving selecting and matching resources from a resource pool with specific requirements. Semantic modelling and searching technologies are commonly used for these kinds of model; the specific semantic model for describing virtual infrastructure, in particular network topologies, is important in this context. Ghijsen *et al.* [11] describe a semantic web based description language for virtual resources and network known as the Infrastructure Network Description Language (INDL), based on the Network Modelling Language (NML). For search, semantic matching and optimisation technologies such as genetic algorithms and Ant Colony Optimisation (ACO) have been explored extensively [12].
2) **Advanced network services** providing applications extra opportunities to optimise data communication. Software Defined Networking (SDN) protocols such as OpenFlow [13] and Network Service Interface (NSI) [4] have attracted substantial attention from both industry and academia. Compared to purely network level protocol optimisation, such as multiple path TCP, these SDN technologies allow applications 1) to customize network connectivity between services by defining suitable flow forwarding tables, or by reserving dedicated links, 2) to virtualize the network resources for different partition schemas by tuning the network slice for given set of computing and storage nodes, and 3) to control the network quality of service by either advanced reservation of links or dynamically controlling the packet flows. However, including these new features in data delivery services is still at a very early stage.
3) **Service Level Agreement (SLA) issuing and real-time negotiation technologies** are needed, but depend heavily on the complexity of the mapping between application requirements and the available resources, and the matching among quality requirements at different service layers. Most mapping approaches are based on graph mapping using key quality parameters such as execution time; however limited

[5] http://www.askalon.org/

[6] www.egi.eu

association between the application and infrastructure during application development makes the searching procedure over large resource graphs very time consuming. In this context, the main approach currently taken to improve the search procedure is to include different types of heuristics and optimization technologies, for instance parallelizing the searching procedure for matching resources and applications [14], pre-processing the resource information by clustering the resource information based on the SLA request, and multi objective optimisation for searching alternative solutions [15].

The SWITCH workbench will implement these components. Scalable cloud facilities provide an attractive platform for deployment of time-critical applications, due (for example) to the cost-effectiveness of renting the necessary resources and acquiring additional resources temporarily, on demand. However, full infrastructure programmability for an individual application is not readily available to application developers at present. SWITCH aims to address this need.

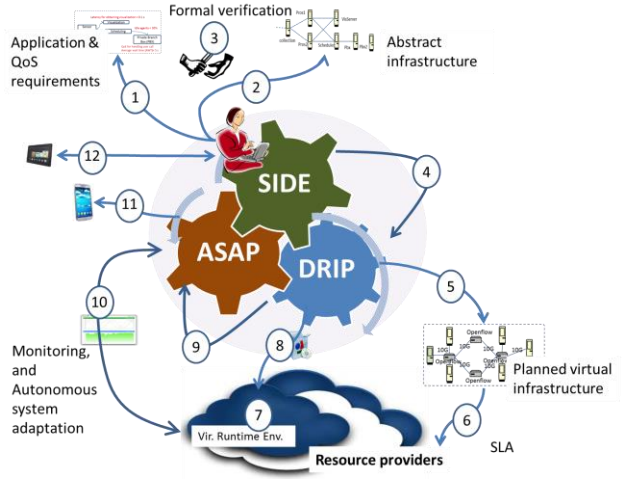

Figure 1: SWITCH Application Lifecycle

It is creating a new software methodology that can improve the development and execution of time-critical applications in Clouds, realised in a workbench which will present the developer with appropriate abstractions to specify his or her requirements and support the entire time-critical application development life-cycle.

SWITCH comprises three sub-systems, illustrated in:

- The SWITCH Interactive Development Environment (SIDE) subsystem provides interfaces for all user- and programmer-facing tools, by exposing Web-orientated graphical interfaces and APIs. SIDE will use HTML5 for the front end, providing interactivity and end-user device portability, and separation from the back-end Web services (hooks) and APIs constructed using Python and tools such as Django, Flask, or similar.
- The Dynamic Real-time Infrastructure Planner (DRIP) subsystem prepares execution of applications developed in the SIDE subsystem.
- The Autonomous System Adaptation Platform (ASAP) monitors the status of the application and adapts the application and run-time environment to maintain compliance with DRIP Service-Level Agreements (SLAs) in a resource-efficient manner.

To implement the above, and to ensure the workbench achieves the usability aims, is a major undertaking. An abstraction layer is required for all components and elements comprising the Cloud application; SIDE will expose them to the developer.

Figure 1 shows the sequence of operations for a typical SWITCH application lifecycle. The developer begins by composing the application logic and defining QoS constraints (step 1). They can also give an abstract network overlay and other infrastructure requirements (step 2). These activities are supported by a formal reasoning component (step 3). This information is passed from SIDE to DRIP, along with requirements such as resource providers to be used and total application execution cost budget (step 4). DRIP plans the concrete virtual runtime environment (computing, storage, network), reasoning with application-level QoS constraints (step 5). It generates Service Level Agreements (SLAs) with the resource provider(s) (step 6), and the virtual environment is provisioned (step 7). DRIP customises the virtual environment and deploys necessary services for the application (step 8), then unbundles and executes the application (step 9) onto a specific computing infrastructure. At run-time, SIDE allows the user to query and visualise the application, run-time environment status and real-time monitoring information (step 10); receive notification of important system status changes (step 11), and directly manipulate system execution (step 12). ASAP also diagnoses system performance, and, where possible, autonomously makes decisions on control actions to restore performance, learning from past execution history in order to refine the decision making procedure (step 10 again).

Many of the technologies used in the WZeroRPC-Sentinel prototype that we present in the following sections, including WZeroRPC, can potentially be used in SWITCH to realise the lifecycle described above. The present prototype is primarily focused on the redeployment capabilities that will be needed within the DRIP subsystem. There is also a simple implementation of the ASAP optimization elements present, but this has not been the main focus of the prototype.

4. WZeroRPC

Despite the modularity of existing workflow engines, described in Section 2, workflow systems generally provide a language to model the entire process and logic defining the interaction of components. Therefore, dynamic interactions happen within the workflow language and structure of the workflow systems, which means that in order to create a dynamic data workflow, a developer would:

- Need to learn to program using the workflow language of a system.
- Be confined to what features that language provides, which in general are never comparable to a conventional programming language

WZeroRPC is an open source platform that takes a different approach by offering a workflow-based execution model that can be embedded within conventional programming paradigms. Specifically, it provides support for executing a directed acyclic graph (DAG) workflow, executed across a distribution of workers, along with the dependencies between them, whilst providing a Python interface for coordinating execution and data dependencies between multiple workflow instances. The DAG aspect provides the workhorse for the raw computation of executing massively parallel workflows and orchestration scenarios, while the Python interface allows Python programmers

to quickly embed workflows into their applications, without learning a new language. It facilitates dynamic data dependent scenarios through the logic of a powerful programming language designed for such a task, i.e. Python. This allows for extremely complex dynamic interactions between multiple concurrent workflow instances.

There are three core services offered by the system: workflow execution, discovery of workers and data management. For execution, WZeroRPC has a centralized workflow manager node, which coordinates the distribution of workflow tasks to a collection of workers, and respects the dependencies between them. The DAG workflows are specified using the Python NetworkX[7] package. They use a special WZeroRPC node object to provide an interface from a NetworkX graph nodes and the underling execution model. All remote invocations are executed using the ZeroRPC package, which a lightweight, reliable and language-agnostic library for distributed communication between server-side processes. Dependencies are control-based dependencies that are topologically sorted and executed in that order to respect the flow between the nodes. Data dependencies between nodes can also be specified by using special tags that allow outputs of one node (elements of the return tuple from an RPC call) to the input of another (as an RPC argument). A similar tagging mechanism is also used to interface with data from a running workflow instance. Multiple workflow instances are managed by a workflow manager, which uses a special plugin to inject customized Python code for manipulating workflow instances in the way the programmer chooses. In this way, a programmer can create workflows, obtain workflow IDs and interface with them whilst reusing the WZeroRPC infrastructure for execution and visualization of the workflows.

For the mapping of tasks to nodes, the system uses a worker model that is managed using a worker pool, to reference the worker instances, and a scheduler that maps between Node instances and the worker that is used to execute the node's work. The worker pool allows workers to register themselves with the system using the discovery subsystem to provide their location and facilities. Once registered, the worker pool instantiates a worker client to connect to the remote service. A factory is provided to instantiate any underlying discovery implementation so that different discovery methods can be plugged into the framework – currently an RPC-based discovery system is implemented for this purpose.

Workers are implemented using a client side and server side to create a channel for each connection. The abstraction is a Python method but this can be connected to any pipe that supports this request/response paradigm. Workers currently have two implementations: RPC for remote method invocation and local for invoking the method locally. Thus, any node in the workflow can be executed locally or remotely using ZeroRPC.

5. SENTINEL WORKFLOW IMPLEMENTATION

The Sentinel ("**Se**ma**nt**ic **Int**e**ll**igence") platform is a social media research and analytics tool that supports semantic enrichment of streamed social media data for the purposes of situation assessment. Figure 2 shows the conceptual stack that belongs to the Sentinel platform which is founded upon a knowledge-based approach, in which input streams (channels) are characterized by spatial and terminological parameters, significant terms and ontology-based tags. This supports interpretation of processed media in terms of the 5W framework (who, what, when, where, why), with the derived information made available through a series of APIs for use in front end applications.

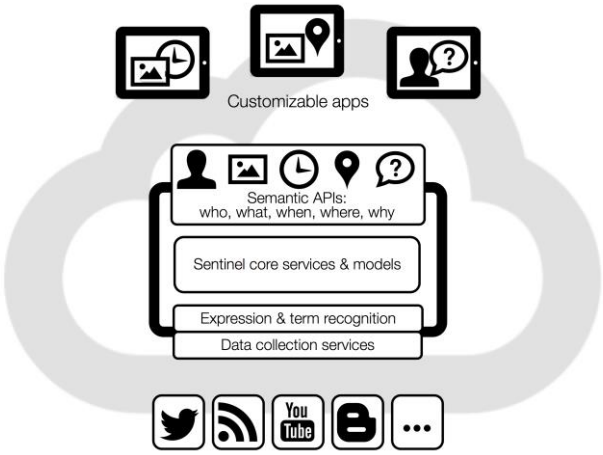

Figure 2: The Sentinel platform conceptual architecture

The Sentinel platform represents a scalable time-critical application, which is highly suitable for cloud deployment, and so was selected as a study case for the WZeroRPC infrastructure.

5.1 Sentinel Use Case

The backbone of the Sentinel platform is its pipeline – a collection of cloud services that can hook into social media APIs (bottom of Figure 2), collect and process the data, and store the vast amounts of information gleaned from these data. The pipeline is modular in nature, allowing for the incorporation of new processing modules, and is accessible via a set of user-facing "apps".

The first module incorporated into the Sentinel pipeline performs recognition of terms (i.e. textual representations of important concepts) by utilizing the FlexiTerm software package. FlexiTerm implements an algorithm for automatic term recognition based on their structure, frequency and collocational stability [16]. It was designed to be robust with respect to linguistic variations seen in social media.

The Sentinel pipeline is designed to be scalable, with the ability for modules to be duplicated in order to share the workload generated by the throughput of social media data. The duplication of modules is supported through the use of an Advanced Message Queuing Protocol (AMQP) service called RabbitMQ. AMQP allows modules to subscribe and publish to shared message queues and hence the social media data, in the form of messages, can be distributed or replicated to multiple modules.

The invocation of modules in the Sentinel pipeline (both the initial copy and duplications) is a manual method, with the need for a script to be run in order to start up and configure a module. It is highly desirable that this startup and scaling of the system be automated.

In order to perform the required monitoring and scaling of the Sentinel pipeline, we defined a number of services that would form part of the prototype system:

[7] https://networkx.github.io/

- **Monitoring Service:** This service will monitor application metrics and decide when the application is in need of scaling.
- **Optimization Service:** This service will make decisions as to how much vertical and horizontal scaling is required for each component in the application.
- **Reconfiguration Service:** This service will perform the reconfiguration of the application rescaling any of the applications as directed by the Optimizer.

The following sections outline the design methods that were undertaken in creating an automatically scalable version of the Sentinel pipeline.

5.2 Identifying Application Metrics & Scenarios

The Sentinel pipeline operates on the concept of a number of "Channels" that consist of a set of search terms relating to a particular topic, each channel has a unique ID for identification within the pipeline. It is the creation and expansion of these channels that drives the amount of data passing through the pipeline.

Initial development of the Sentinel platform and pipeline has used Twitter as the primary driver for Sentinel since tweets have the characteristic of frequently being used as "link carriers" to the other forms of social media, for example, a tweeted link to a news story, a YouTube video, or a blog posting. Therefore we continue with the use of Twitter as the main data source within the workflow scenarios.

Table 1: Reconfiguration Scenarios

Scenario	Description
Scenario 1	A new Channel is added to the Pipeline (The end users have decided to start investigating a new subject)
Scenario 2	A Channel increases in size (The end users either have decided to expand the search set for an existing subject, or the search terms already being used have started to bring back more results than in the past).

Table 1 lists the two scenarios that can influence the rate at which data is processed within the Pipeline, both of which are driven by changes in the collection channels.

Table 2 lists measurable application metrics that can be retrieved from pipeline modules, which will help identify when channels have been created (both CHS and TPS increase), when a channel has increased in volume (both TPS/CH and TPS increase), or when storage services are reaching capacity (the increase in TACCUM).

Table 2: Sentinel Metrics for Determining Application QoS

Metric	Meaning
TPS	Total incoming Tweets per second rate within the Pipeline
CHS	The number of Channels running in the Pipeline
TPS/CH	Total incoming Tweets per second on a specific Channel
TACCUM	Tweets Accumulated by the Pipeline

5.3 Identifying Scaling Modes

Modules within the pipeline relate to a particular channel in different ways, and so it was important to identify the different relationships that each module has with respect to Channels within the Pipeline.

There are two channel exclusive modules that have a direct one-to-one relationship between Channels and processes:

- Twitter Collection manages API requests to Twitter and streams back Tweets.
- Aggregator Manager schedules Tweet aggregation jobs for the Channel.

There is currently one channel acquirable module; FlexiTerm which will be reserved to processing a single Channel's data periodically. The remaining modules are channel agnostic, meaning they can process data from any Channel at any time.

As well as the modules, the services within the Pipeline may also be scaled to cope with the throughput of the application.

We assigned each of these behavioral groups a colour (channel exclusive: Blue, channel agnostic: Green, channel acquirable: Orange, services: Yellow) and then identified which of the application metrics will influence whether a component group would require horizontal or vertical scaling. This is shown in Table 3 below.

Table 3: Behavioral Group Application Metrics

Component Group	Horizontal Scale Modes	Vertical Scale Modes	State
Blue	CHS	TPS/CH	YES
Green	TPS	-	NO
Orange	CHS	TPS/CH	NO
Yellow	-	TPS + TACCUM	NO

Both the Blue and Orange groups are influenced by channel specific metrics, with the distinction between the two groups being that components in the Blue group are stateful; being bound to a particular channel and thus requiring a channel ID.

5.4 Reconfigurable Infrastructure

The current deployment of Sentinel consists of Docker containers, running within a CoreOS deployment, housed on an OpenStack deployment. OpenStack is a cloud computing software platform, providing IaaS services. The Juno release is deployed for this configuration. This deployment is backed by Neutron OpenVSwitch networking, and Ceph distributed storage. The CoreOS deployment consists of 12 virtual machines running CoreOS. Two of these VMs have 2 vCPUs, 4GB RAM and 160GB storage, the rest have 4 vCPUs, 6GB RAM, and 160GB storage. CoreOS provides an init-system style abstraction to the cluster of compute resources, heavily utilizing SystemD. To provide this abstraction, along with SystemD, it utilizes a distributed configuration store called ETCd, and wraps everything up an in abstraction layer called Fleet. On top of this, the containers for Sentinel are deployed. Fleet provides many of the features that SystemD provide on a cluster basis, such as defining before/after requirements, to ensure a needed service is already running before running a component that requires it. The

Sentinel containers are aware of the ETCd configuration store, and utilize it to obtain initial configuration data.

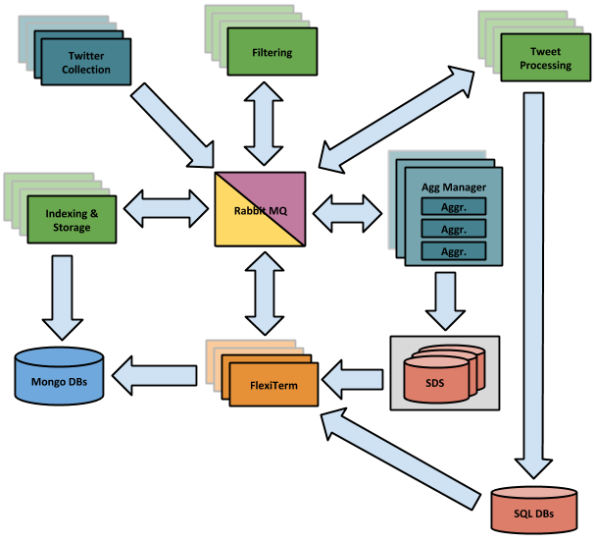

Figure 3: Distributed Architecture of the Sentinel Pipeline

To deploy Sentinel onto CoreOS, currently a simple script is run, that populates ETCd with the necessary configuration values, uploads the units defining each service to Fleet, and then starts them one by one. This script is very rigid, and suited only to one particular deployment, and would require heavy modification to handle a new deployment. The intention for the design of this system was to create a "glass box" platform, which allows for transparency in operations, along with facilitating platform extensibility both internally and externally. This is in opposition to the many existing "black box" solutions. Being distributed, it is well suited to the Cloud/IaaS environment, deployable not only across multiple nodes, but across different environments too. Each part of Sentinel was designed in a modular fashion, allowing components to be "dropped in", moved around, and replaced, with minimal to no effect on the rest of the system.

Figure 3 shows all the components present within the Sentinel Pipeline and where data is passed between them. Components were designed to be as decoupled from each other as possible, and interaction between components is achieved using RabbitMQ. The majority of the other components are processes that consume and produce messages in order to perform data-driven topic identification on Social Media. These processes are modular and cover everything from data collection, filtering, sanitation and processing. The remainder are other services within the Pipeline, primarily data storage (SQL, NFS and MongoDB).

All processes and services present in the Sentinel Pipeline have been wrapped in Docker containers allowing for the processes to be easily redistributed on cloud infrastructures. Both horizontal scaling (the replication of process/service containers) and vertical scaling (increasing a container's resources such as CPU, RAM or Disk space) are achievable.

5.5 WZeroRPC Monitoring and Dynamic Management

After identifying how component groups may scale, dependent on the application metrics, we focused on the reconfiguration workflow that is necessary to reinitialize the Pipeline components when the need for any form of application scaling arises.

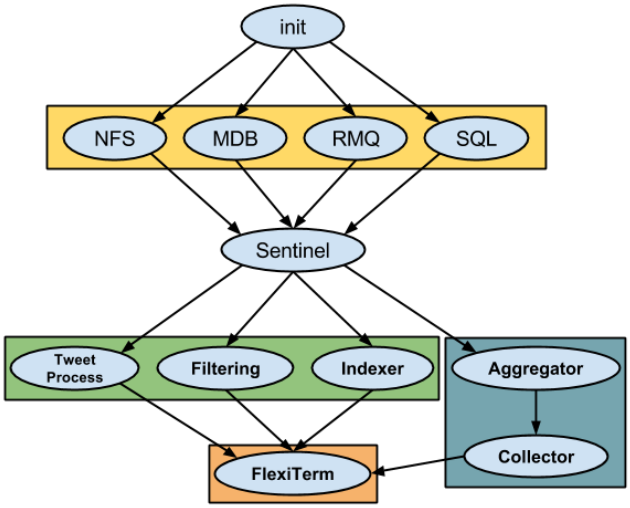

Figure 4: Sentinel Pipeline Reconfiguration Workflow

The reconfiguration workflow is shown in Figure 4. It is derived from the dependencies between component types. With all the processing components being dependent on the service components, the *init* node is used to initialize the reconfiguration nodes of all service components (Yellow component group) first. The *Sentinel* node listens for all the service nodes to complete and then initializes the reconfiguration of all Green group component nodes and the Aggregator component nodes. The dependency between the Blue group components exists on a channel by channel basis, as the Aggregator must be ready and waiting to receive data before a particular channel's Collector is initialised, and so the reconfiguration workflow recognises that. Finally the *Flexiterm* node listens for the completion of all Green and Blue nodes before kicking off Flexiterm reconfiguration.

In the WZeroRPC implementation, the above design was split into three different workflows to focus on the different modes of scaling that are required: horizontal; vertical and when both horizontal and vertical scaling is required simultaneously.

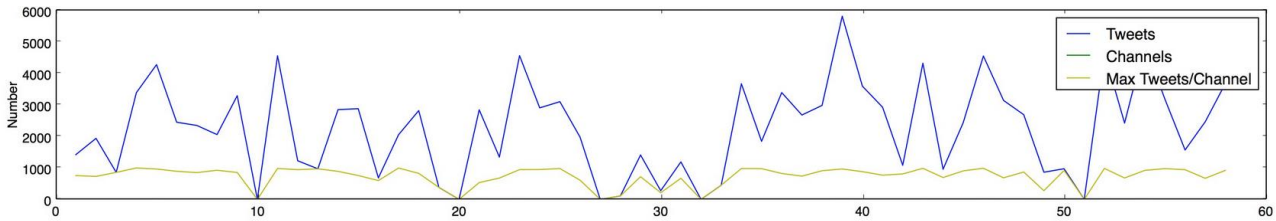

Figure 5: WZeroRPC Monitoring Output from Sentinel Pipeline Metrics

In the WZeroRPC implementation, there is also a monitoring workflow that queries Sentinel's parameters and passes these to an Optimiser component, which analyses the current throughput rates and Channels and calculates whether any reconfiguration is necessary. In the current prototype this scaling requirements are identified using a decision tree based upon metric thresholds the application designers. The monitoring output is displayed in the WZeroRPC GUI, as illustrated by Figure 5, along with the currently active workflow which are collectively presented in Figure 6, with Figure 6a showing the initial monitoring workflow that continuously runs until scaling is required.

In order for the Optimizer to inform the Reconfiguration Service workflows of the required scaling, a reconfiguration payload must be provided. The configuration payload can be "diffed" against the previous configuration to determine what changes must be made to the application (and thus, from where on the Reconfiguration Workflow action must begin). This payload consists of a list of all the containers that need to be deployed for the application. Any horizontal scaling up or down is identified here by addition or removal of rows in the payload.

Each container is listed along with a number of configuration parameters either defined by the Application state (STATE) or by the Monitoring Service (MON), the latter being the means of defining any vertical scaling required. The HARD and SOFT limits define whether the parameter is NULLABLE or not. The list of these parameters is shown in Table 4 below.

Table 4: Application and Container Parameters

Param Type	Container Parameter	Range	Limits
MON	CPU	1 - 2	HARD
MON	RAM	1 - 4	HARD
MON	DISK	40 - 80	SOFT
MON	NET	1 - 8	HARD
STATE	CHANNEL ID	#	SOFT
STATE	CHANNEL ID STATE	ON - OFF	HARD
STATE	SCALING	Scaling Modes (a set)	HARD
STATE	FUNCT	TEXT/TAGS	HARD
STATE	IMPL	LANGUAGE	SOFT
STATE	OS	OPERATING SYSTEM	SOFT

Once the Optimizer has processed this information, it will then pass the reconfiguration data payload to the main WZeroRPC coordinator. The coordinator will then check the payload to find out if Sentinel requires reconfiguration and if so, what type of reconfiguration it requires. It then invokes one of the three workflows to reconfigure the infrastructure to scale horizontally, vertically or both.

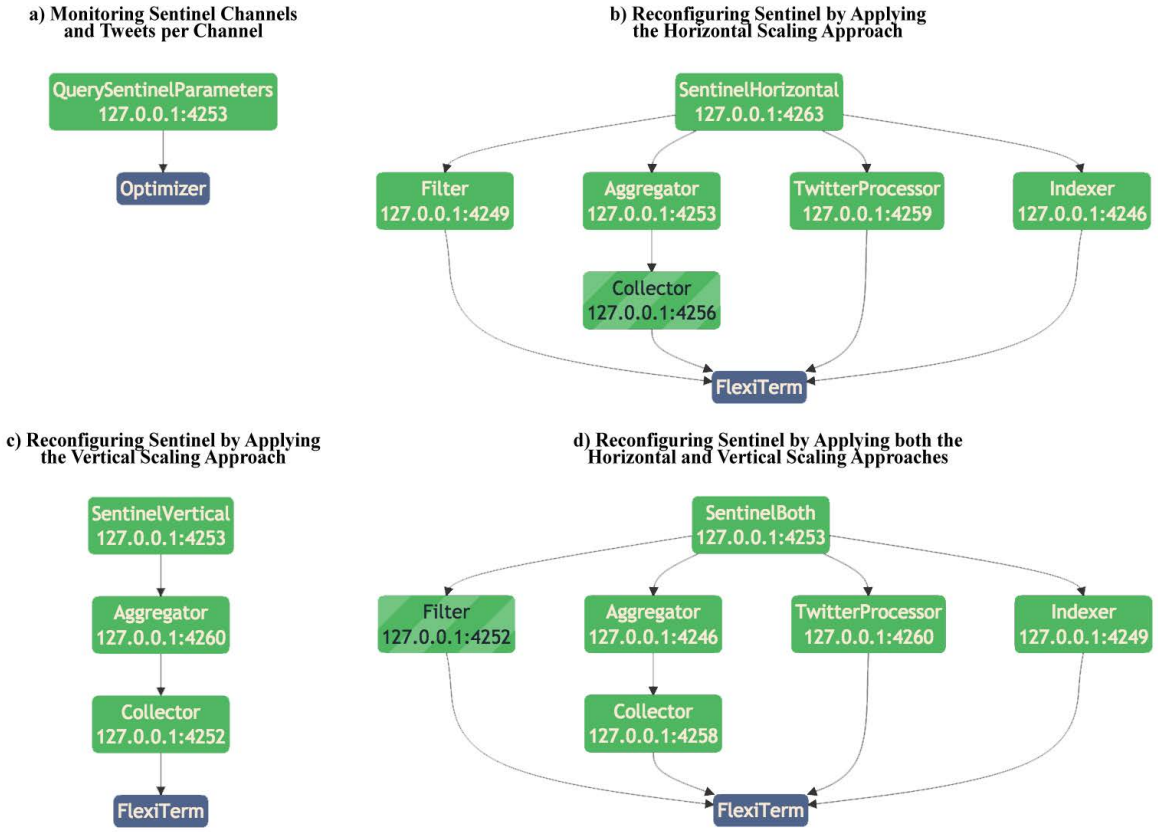

Figure 6: WZeroRPC Control Workflows Being Used to Reconfigure the Sentinel Pipeline

Figure 6b shows the horizontal reconfiguration workflow. The payload is first processed by the SentinelHorizontal RPC method, which synchronizes and coordinates the reconfiguration order. It first sends instructions concurrently to the Filter, Aggregator, TwitterProcessor and Indexer reconfiguration components to instruct them to reconfigure. Once the Aggregator has been reconfigured the Collector processor is then reconfigured to change the collection algorithm. The Aggregator must be reconfigured before the Collector so that it has the appropriate number of Docker containers to process the new Channels that have entered the network. Then, once all of these components have been reconfigured, the FlexiTerm component is finally reconfigured to complete the process.

In Figure 6c, Sentinel is reconfigured vertically. This workflow is far simpler because there are fewer components that should expand vertically as the throughput requirements increase. Since FlexiTerm extracts correlations in the data it must be capable of processing all of the tweets in one channel. Therefore as those tweets increase, e.g. during a football game, there will be more tweets about that game, and so it must increase its power and memory to be capable of processing the new data sets. Therefore, in this pipeline, the Aggregator is reconfigured, followed by the Collector, and finally FlexiTerm is reconfigured to complete the process.

Figure 6d shows the combination of both the horizontal and vertical scaling approaches. This workflow is identical to the horizontal workflow except that the Optimiser payload provides the necessary data to instruct each component to scale vertically and horizontally.

6. Further SWITCH Use Cases

SWITCH will also be using a number of other time critical applications within the project: (i) Collaborative real-time business communication platform; (ii) Elastic disaster early warning system, and (iii) Cloud studio for directing and broadcasting live events. These additional use cases provide a broad range of challenges and considerations that will need to be incorporated into the process, when onboarding applications into a Workflow supported reconfiguration and scaling environment. Indeed, all these scenarios have time-critical aspects, and it is important to ensure that Quality of *Service* (QoS) requirements are enforced (e.g. to ensure appropriate bandwidth).

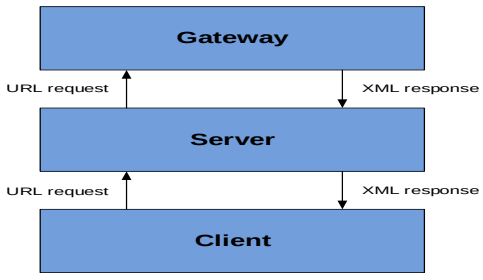

Figure 7: Communication between Telemetry System Components

For example, the early warning telemetry application uses clients, servers and gateways to coordinate sensors distributed across a geographic feature, as presented in Figure 7. The telemetry server handles various kinds of information grouped together in entities called nodes, with communication being handled via A2A ("addVANTAGE-to-ASCII") [17]. Each node has an ID that must be unique on this server. Thus, when requesting the telemetry server using a node ID, this node (and its subnodes – depending on the request) is used for the response or none if this ID is unknown. The client, on the other hand, can import nodes, giving them a new ID that is unique locally, but also it needs to keep the ID on the telemetry server, called the remote ID.

7. CONCLUSION AND FUTURE WORK

This paper describes a workflow implementation of infrastructure reconfiguration tasks and dependencies for time critical applications. The Sentinel application, which processes tweet data from social media networks, provides a good example of an application that requires on demand reconfiguration of its components based on the current processing requirements of tweet Channels. Each Channel represents a semantic search and therefore is processed independently of other channels. Therefore, multiple Channels can exist concurrently and also channels can grow rapidly as the interest in a particular topic increases by the users of the network. Therefore, Sentinel requires a combination of horizontal and vertical scaling to combat both of these processing throughput conditions. We showed that WZeroRPC can be used to provide the necessary dependencies to automate the reconfiguration of the components and infrastructure in a lightweight way.

Although the WZeroRPC-Sentinel prototype demonstrates many of the principles of achieving adaptability for time critical applications in a cloud environment, it still does not make full use of recent advances in virtualized infrastructure.

The work described here therefore can be considered as an initial prototype of the implementation to demonstrate that the SWITCH project's goals can be met using dynamic workflows. Focus has been made on the reconfiguration of applications that is undertaken by the DRIP subsystem. In the future, we expect to refine this process to provide finer granularity within the decision making to determine how and when components are scaled. This will be developed into a sub component of ASAP subsystem. This would provide far more complex workflows and decision-making and would be capable of implementing more flexible reconfiguration modes. The implementation here however proves that the concept can work and is currently being used to experiment with the reconfiguration of Sentinel during real-time feeds.

ACKNOWLEDGMENTS

The WZeroRPC framework was collaboratively developed by Ian Taylor, Joe Macker, and Brian Adamson. The SWITCH project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 643963. Development of the Sentinel platform was funded by the European Commission under the project "Tackling Radicalisation in Dispersed Societies (TaRDiS)", and the ESRC via the project "After Woolwich: Social Reactions on Social Media" (ES/L008181/1). The Sentinel platform was co-designed by experts in policing and neighbourhood security at Cardiff Universities Police Science Institute, http://www.upsi.org.uk. We are particularly grateful to Martin Innes, Colin Roberts and Sarah Tucker for their insights.

REFERENCES

[1] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, pp. 528-540, 2009.

[2] S. Gesing, M. Atkinson, R. Filgueira, I. Taylor, A. Jones, V. Stankovski, *et al.*, "Workflows in a dashboard: a new generation of usability," in *Workflows in Support of Large-Scale Science (WORKS), 2014 9th Workshop on*, 2014, pp. 82-93.

[3] C. Roberts, M. Innes, A. Preece, and I. Spasic, "Soft facts and spontaneous community mobilisation: the role of rumour after major crime events," *Data for Good: How big and open data can be used for the common good, P. Baeck, ed*, pp. 37-43, 2015.

[4] Z. Zhao, C. Dumitru, P. Grosso, and C. De Laat, "Network resource control for data intensive applications in heterogeneous infrastructures," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, 2012, pp. 2069-2076.

[5] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, *et al.*, "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud," *Nucleic acids research*, p. gkt328, 2013.

[6] I. Taylor, M. Shields, I. Wang, and A. Harrison, "The triana workflow environment: Architecture and applications," in *Workflows for e-Science*, ed: Springer, 2007, pp. 320-339.

[7] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, *et al.*, "Pegasus: Mapping scientific workflows onto the grid," in *Grid Computing*, 2004, pp. 11-20.

[8] M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of scientific workflows in the ASKALON grid environment," *ACM SIGMOD Record*, vol. 34, pp. 56-62, 2005.

[9] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec, "Flexible and efficient workflow deployment of data-intensive applications on grids with MOTEUR," *International Journal of High Performance Computing Applications*, vol. 22, pp. 347-360, 2008.

[10] D. Meyer, "The software-defined-networking research group," *Internet Computing, IEEE*, vol. 17, pp. 84-87, 2013.

[11] M. Ghijsen, J. Van Der Ham, P. Grosso, C. Dumitru, H. Zhu, Z. Zhao, *et al.*, "A semantic-web approach for modeling computing infrastructures," *Computers & Electrical Engineering*, vol. 39, pp. 2553-2565, 2013.

[12] W. Dong, Y. Liu, C. Chen, J. Bu, C. Huang, and Z. Zhao, "R2: Incremental reprogramming using relocatable code in networked embedded systems," *Computers, IEEE Transactions on*, vol. 62, pp. 1837-1849, 2013.

[13] H. Jin, D. Pan, J. Liu, and N. Pissinou, "OpenFlow-Based Flow-Level Bandwidth Provisioning for CICQ Switches," *Computers, IEEE Transactions on*, vol. 62, pp. 1799-1812, 2013.

[14] C. Muller, M. Oriol, X. Franch, J. Marco, M. Resinas, A. Ruiz-Cortés, *et al.*, "Comprehensive explanation of SLA violations at runtime," *Services Computing, IEEE Transactions on*, vol. 7, pp. 168-183, 2014.

[15] A.-F. Antonescu, A.-M. Oprescu, Y. Demchenko, C. De Laat, and T. Braun, "Dynamic optimization of SLA-based services scaling rules," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, 2013, pp. 282-289.

[16] I. Spasić, M. Greenwood, A. Preece, N. Francis, and G. Elwyn, "FlexiTerm: a flexible term recognition method," *Journal of biomedical semantics*, vol. 4, p. 27, 2013.

[17] A. Ochian, G. Suciu, O. Fratu, and V. Suciu, "Big data search for environmental telemetry," in *Communications and Networking (BlackSeaCom), 2014 IEEE International Black Sea Conference on*, 2014, pp. 182-184.