StreamMine3G OneClick
Deploy & Monitor ESP Applications With A Single Click

Andrey Brito†, André Martin*, Christof Fetzer*, Isabelly Rocha†, Telles Nóbrega†

*Technische Universität Dresden - Dresden, Germany - Email: {firstname.lastname}@se.inf.tu-dresden.de
†Universidade Federal de Campina Grande - Campina Grande, Brazil - Email: {andrey, isabellylr, tellesmvn}@lsd.ufcg.edu.br

Abstract—In this paper, we present StreamMine3G OneClick, a web based application relieving users from the burden of the complex installation and setup process of Event Stream Processing (ESP) systems such as StreamMine3G in cloud environments. Using StreamMine3G OneClick, users just upload their application logic and choose their favorite cloud provider such as Amazon AWS. StreamMine3G OneClick performs an automatic installation process ensuring a correct setup of StreamMine3G and dependent services such as Zookeeper in the cloud. Furthermore, StreamMine3G OneClick offers a convenient graphical monitoring interface giving users an instantaneous insight of the performance to identify bottlenecks as well as to easily trace down bugs in their applications.

I. INTRODUCTION

During the last decade, we have been witnessing of a massive growth of data. Technological advances allow more and more types of electronic devices to be connected to the Internet and hence act as new data sources for various applications and the creation of new businesses models. Application examples range from processing data from mobile devices for location-based services to the monitoring and control of large critical distributed systems like the power grid. In the first example, smartphone data can be used to generate periodic reports of the geographical location of their owners and create content for social networks, or to receive notifications for opportunities such as friends in the neighborhood or shopping opportunities. In the context of smart societies, smart meters deliver energy consumption data in real time to energy providers allowing for a better demand prediction and early detection of problems in a smart grid.

The advent of these new types of data sources calls for a new way of processing data and hence motivated the development of Event Stream Processing (ESP) systems, which aim at processing data in (soft) real time and in large scale to detect situations of interest. In fact, ESP describes a whole set of techniques to address such classes of problems [1], [2]. Contrary to MapReduce systems such as Hadoop [3], which follow a store-and-process pattern, ESP systems target the processing of large amounts of data with low latencies. Besides producing results more quickly and allowing immediate reaction to relevant situations, ESP systems are also well suited in scenarios where more data is being generated than can be stored. In this case, real time processing enables data to be summarized and only relevant data need to be stored.

During the past years, many open source as well as commercial ESP systems and frameworks have been developed and reached a high level in maturity. However, with increasing feature set size, the complexity of those applications and of their deployment has increased. As an example, Storm, a popular open source ESP tool used in major sites such as Twitter [4], relies on several third party libraries such the JVM, Zookeeper (a coordination service for distributed systems [5]), Nimbus and zeroMQ [6]. Writing an application, i.e., designing a new operator for such a system and running it in a non-distributed manner locally is simple as comprehensive documentation and examples exist. Nevertheless, getting such a system set up properly in a distributed manner requires the installation and configuration of many third party libraries and hence puts a heavy burden on the hands of the application developer.

Once the developer succeeded in setting up the application in a distributed manner by using locally available machines or VMs, for example, another layer of complexity is added when the application is required to be run in cloud environments using on demand resources such as Amazon EC2. Since there is no standardized API for acquiring, configuring and releasing cloud resources, the developer is required to juggle around with different sets of tools.

Identifying performance bottlenecks in a distributed systems is another well known challenge and event processing applications add another level of complexity as events are continuously traversing a complex graph of operators, performing concurrent state modifications. Essential for tracing down the causes of a performance bottleneck is the monitoring of the system as a whole as well as its individual parts. A vast amount of cluster monitoring systems and tools such as Nagios and Ganglia exist, covering the system aspect of monitoring, however, it is equally important to understand the behavior of individual operators by tracing and monitoring the middleware and application metrics in
the system.

In this paper, we present the design and implementation of StreamMine3G One Click. StreamMine3G OneClick is an intuitive web application that relieves the developer from the burden of installing and setting up an ESP system such as StreamMine3G on cloud environments. In addition to the installation process, the system manages automatic acquisition as well as deallocation of resources depending on the utilization of the cluster. Furthermore, the web application offers an instantaneous view on all processing elements of the system to track down performance issues as well as bugs in operators.

The rest of the paper is organized as following. In Section II we provide an overview about the cloud computing model, ESP systems in general, and StreamMine3G, our ESP system. In Section III we detail the StreamMine3G OneClick system architecture and its individual components and discuss related toolsets. We then present a short evaluation of our approach in Section IV. The paper concludes with Section VI, which summarizes our ideas and contribution.

II. BACKGROUND

In this section, we provide some background information on the cloud computing model, ESP systems in general, and StreamMine3G, our ESP system.

A. Cloud Computing

Cloud computing has become an attractive alternative to many small to medium sized companies as cloud resources can be dynamically acquired (and released) as needed and hence reflect almost naturally the growth of a company. One of the most popular cloud providers is Amazon AWS with its EC2 product, providing the acquisition of cloud resources either at a fixed per hour pricing model or via a bid market using so-called spot instances.

In the context of ESP applications, the cloud computing model offers a new way of coping with load fluctuations as new nodes can be dynamically acquired, added to the compute cluster to split and redirect event streams to the newly acquired resource and hence balancing the load as the incoming data rate increases.

In addition to coping with load fluctuations due to increasing input data rates, the cloud computing model also enables the ad-hoc deployment of new queries for a more in-depth data analysis of some event stream as new computational resources can be acquired at any time.

B. ESP Systems

ESP is a well-known set of techniques for processing streams of data with low latency. Applications written for ESP systems typically consist of a set of operators connected as an acyclic graph. Events then traverse such operator graph and useful information is derived, for example, through filtering or aggregation of the events.

Some ESP systems (ESPER [7]) are equipped with a rich sets of standard Complex Event Processing (CEP) operators along with a SQL-like query language to express how information should be extracted. Systems such as Storm [4] or Apache S4 [8] provide a more flexible MapReduce-like interface enabling the application developer to write custom operators for data extraction.

In contrast to the classical MapReduce, operators in ESP systems operate on windows of events (sliding or jumping) and hence are stateful. Keeping state is necessary for implementing operators that execute some form of aggregation. For example, frequency estimation (top-k), pattern detection and matching, and moving averages are common operations in ESP applications.

However, operator state imposes new challenges regarding scalability, elasticity, and fault tolerance. In a parallel environment, access to state must be protected against concurrent modifications. Traditional concurrent programming (e.g., with locks) is not trivial and does not scale enough to harness nowadays multi-core architectures. Combining the MapReduce approach with ESP enables state to be partitioned and both vertical and horizontal scaling to be achieved. Furthermore, for elasticity, operators (including their state) may have to be transferred to new machines. Operator migration imposes the challenge of guaranteeing consistency during migrations so no duplicate events are processed or events lost. Similarly, it is also a challenge to enable an ESP system to recover after a fault.

C. StreamMine3G

In this section, we outline the design and architecture of StreamMine3G, our ESP system.

StreamMine3G provides a MapReduce like operator interface for writing custom operators, if desired, and a rich set of state-machine-based standard CEP operators [9] with a CQL like query language interface for users familiar with database tools. StreamMine3G supports stateful operators and, this, an operator can create, access, and modify state. As state in StreamMine3G is partitioned using keys, events that map to different pieces of the state can be processed in parallel. However, events accessing the same portion of the state are serialized to ensure consistency.

Once a set of operators have been designed, developers need to define the operator topology, i.e., an acyclic graph of operators. If the developer opted for standard CEP operators in conjunction with the CQL interface, the topology is defined implicitly.

StreamMine3G is a highly scalable and elastic ESP system. Scalability is achieved through the partitioning of operators (similar to MapReduce) to harness the processing capability of both cores of a machine, and of machines in a cluster. A partition of an operator in StreamMine3G is called a slice. In a previous work [10] we have evaluated the scalability of this programming model.
A typical StreamMine3G cluster consists of multiple nodes where a node represents either a physical or a virtual machine running a StreamMine3G process/instance where each StreamMine3G node can host an arbitrary number of slices of arbitrary operators. In order to decide what slice should be running on what StreamMine3G node, StreamMine3G provides a manager interface.

The manager interface offers a rich set of API calls to deploy and undeploy slices on specific StreamMine3G nodes as well as the possibility to migrate a slice from one StreamMine3G node to another node. This is especially useful to shift load off when one node becomes overloaded or if nodes are underutilized. To detect over- or underutilized nodes and take appropriate actions such as triggering a slice migration, the manager receives heartbeat messages from all nodes on a periodic basis. The heartbeats contain information about the current CPU, network, and memory utilization, as well as statistics such as event throughput and state size for each individual slice.

To avoid making the manager component a single point of failure, StreamMine3G relies on Zookeeper [5], a widely used distributed coordination service, to store its configuration. Zookeeper uses replication to avoid loss of information in the presence of faults.

In addition to a fault tolerant manager, StreamMine3G also provides fault tolerance for operators using active [11] and passive replication [12]. StreamMine3G employs a deterministic execution component to ensure consistent processing of events across replicated operators as a less costly alternative to full atomic broadcasts. Fault tolerance based on passive replication is achieved through rollback-recovery using periodic checkpoints and in-memory logging.

III. StreamMine3G OneClick Architecture

We now describe the architecture and provide details about the individual components of our StreamMine3G OneClick platform. We also outline how those components interact with each other.

StreamMine3G OneClick consists basically of the following three components as shown in Figure 1:

- **StreamMine3G OneClick UI** - the user front-end where the user can upload his operator code, define topologies, and have them deployed and run on their favorite cloud provider;
- **Scheduler and Load-balancer** - which employs the manager component responsible for distributing slices across a set of nodes acquired through the infrastructure controller;
- **Infrastructure Controller** - for acquiring and deallocating resources from Cloud providers such as Amazon AWS.

A. StreamMine3G OneClick UI

StreamMine3G OneClick UI is a J2EE web application that can be conveniently accessed with any Internet browser. Once the user logged in, the welcome screen offers four different views:

- **Operator View** - allows the user to register and upload operator code (as a shared library file) for usage in topologies later on. Furthermore, a set of optional parameters for each operator can be provided such as the number of partitions (slices) to define the degree of partitioning and scale-out factor upon deployment as well as if slices of such an operator shall be deployed on a dedicated node or not.
- **Topology View** - provides the user with graphical way to compose topologies, i.e., defining the event flow by assigning up and downstream operators for each individual operator. Furthermore, users can start and stop topologies. Upon start, slices will be deployed using the Scheduler and Load-balancer component on the nodes currently available and removed upon stop.
- **Performance Monitoring View** - enables users to examine performance metrics such as CPU, memory, and network utilization, as well as slice level information such as state size, incoming and outgoing event rate. This view also allows toggling between a topology and infrastructure view. In the former mode, users can identify potential bottlenecks within the operator graph whereas the latter one visualizes the locality within the underlying infrastructure to identify performance issues caused by sharing resources on infrastructure level.
- **Infrastructure View** - allows the user to manually add or remove nodes to the resource pool by launching (requesting) or terminating VMs from a cloud provider of his choice. Note: The Scheduler and Load-balancer component can also performs an automatic acquisition of new nodes as well as releasing idle resources based on the current overall system load if desired.

B. Scheduler and Load-balancer

The Scheduler and Load-balancer component is responsible for deploying and un-deploying topologies on a set of nodes once a request has been received through the StreamMine3G OneClick UI component. Furthermore, the component is constantly monitoring the utilization of the system and reports performance metrics back to the StreamMine3G OneClick UI component for visualization for the developer.

In our current implementation, operator slices of a topology are deployed in round-robin fashion. However, more complex implementations of this component are possible and can be easily replaced. One alternative would be collocating operator slices of CPU intensive operators with slices from IO intensive operators to reach an optimal overall node resource utilization.
Furthermore, this component is directly connected to the Infrastructure Controller Component. This coupling allows the component to take concrete actions such as acquiring new resources if the overall load is exceeding the limit of the current resource pool, i.e., elastically expanding a StreamMine3G cluster as well as contracting the resource pool by migrating and collocating less utilized operators and deallocating the idle resources.

C. Infrastructure Controller

The last component, the Infrastructure Controller handles the acquisition and the release of resources from arbitrary cloud providers as well as installing and configuring individual software components. The implementation of this component encapsulates the non-standardized APIs of various cloud providers available and the complex set of bash scripts for automatically preparing and setting up acquired virtual (or even physical) machines to a StreamMine3G cluster. In our current implementation, we provide cloud adapters for Amazon AWS cloud and for deployment on dedicated clusters.

IV. Evaluation

In this section, we present a micro benchmark we executed using our StreamMine3G OneClick application and the embedded Amazon AWS EC2 connector. The operator topology for this benchmark consists of three operators: A source operator generating events and simulating a continues stream of events coming from some external entity, a worker operator which performs some processing and a sink operator accepting the results from the previous one.

The experiments were executed using 10 micro EC2 instances, each equipped with a single virtual CPU as well as 615 MB of RAM. In order to verify the horizontal scalability of our approach, we partitioned each of the operators using a factor of 3 resulting in a total of 9 slices. Each of the slices has been deployed on a distinct node using our scheduler component. Node 1-3 are hosting the source, node 4-6 the worker and node 7-9 the sink operators respectively.

Figure 2 show the CPU utilization over a course of 30 seconds. node000 hosted the scheduler and load balancer component of our system and hence consumed only little CPU resources whereas the CPU utilization of the remaining nodes hosting the operator slices fluctuates over time and reaches a peak of 70%.

Figure 3 depicts the corresponding throughput measurements for the same experiment. Slices with sliceUId 0-2 represent the source, 3-5 the worker and 6-8 the sink operators, respectively. The throughput in this experiments reaches around 40 $kEvents/s$ peak which matches our expectations as we have been able to reach around 300 $kEvents/s$.
running the same experiment on our dedicated cluster using 8 cores machines.

Contrary to experiments we performed on our dedicated cluster, the throughput and the CPU utilization exhibit high fluctuations. We account the fluctuation to overprovisioning within the Amazon cloud of this instance type (EC2 micro) as it is primarily dedicated for testing purposes and hence might expose naturally this unstable behaviour.

V. RELATED WORK

Data processing in the large has been become very attractive with the advent of Google’s MapReduce [13] and its open source implementation Hadoop [3]. The framework provides a simple and intuitive programming interface and comes with a scheduling component that distributes work across a set of nodes. Furthermore, the framework is equipped with a web interface allowing the user to examine the health status of the nodes and the progress of a previously submitted job. However, setting up such a Hadoop cluster still requires a tremendous effort: Machine need to be equipped with NFS to share Hadoop’s configuration globally as well as operators topologies must be defined in code prior uploading using a command line tool. To add new resources to a Hadoop cluster during runtime for coping with increasing load, bash scripts must be invoked manually. Moreover, the web interface only provides a rough overview about the progress of a submitted Hadoop job, however, does not provide detailed insights about the performance of individual map or reduce tasks as well as peer nodes of a Hadoop cluster.

Amazon AWS offers MapReduce as PaaS (Elastic MapReduce – EMR), relieving the burden from developers installing and setting up a distributed system such as Hadoop similar as with StreamMine3G OneClick. However, EMR is a regular batch processing tool (data to be processed must be first stored in the the S3 service before the job is launched) and cannot be online for continuous processing as ESP tools. In addition, EMR imposes an additional cost over the costs for the EC2 instances actually used.

During the last years, we have been also witnessed a large trend moving from batch processing such as popular Hadoop to (soft) real time processing using event processing systems. Apache S4 [8], a popular and open source event processing platform provides a similar developer friendly programming interface as Hadoop, however, provides (soft) real time processing capabilities in contrast to Hadoop. Although Apache S4 is targeting real time processing, the framework does not provide a web interface with monitoring capabilities. A few projects exists that provide scripts for an deployment of Apache S4 in cloud environments such as Amazon AWS, however still requires a thoroughly understanding of bash as well as manual work.

One system that comes closest to StreamMine3G OneClick is Storm [4]. Storm provides a MapReduce like interface as well as a user interface (Storm UI) enabling users to view metrics such as event throughput for individual operators in a table/text view. Furthermore, several sub projects exist offering bash scripts for a Storm cluster deployment on Amazon AWS. However, the offered components are not fully integrated and provided as a single package leaving again the burden to the developers to read documentation and performing manual installation steps.

VI. CONCLUSION

In this work, we presented StreamMine3G OneClick, a web application that relieves developers from installing, configuring and deploying an ESP system such as StreamMine3G on cloud environments like Amazon AWS. The goal is to make StreamMine3G more approachable for novice users and provide StreamMine3G as PaaS – Platform as a Service, hence allowing developers to entirely focus on their application logic without having to worry about infrastructure and the distributed system setup in the cloud. With StreamMine3G OneClick application developers can program and test their operators locally on their machines and simply upload the operators for the application to be run in the cloud with a few mouse clicks.

StreamMine3G OneClick frees developers from reading long documentations, understanding bash scripts and configuration files to get distributed systems (such as StreamMine3G itself) and dependent services like Zookeeper setup properly. Furthermore, StreamMine3G OneClick allows developers to use local or cloud resources lowering both setup and maintenance costs of dedicated data centers. Identifying bottlenecks is simplified through the visualizing of performance metrics of the system as a whole as well as for individual components through the StreamMine3G OneClick web interface.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community’s Seventh Framework Programme (FP7/2012-2015) under grant agreement number 318809 (LEADS) and from the CAPES, DAAD and GIZ through the NoPa program (TruEGrid project, 2011-2013).

REFERENCES

Throughput (kEvents/s)
CPU utilization (%)

Time (s)

CPU Utilization over a course of 30 seconds using Amazon EC2 micro instances.

Figure 2.

Throughput (kEvents/s)
Processing
Incoming
Outgoing

Time (s)

Event throughput over a course of 30 seconds using an operator graph consisting of a source (0-2), worker (3-5) and sink operator (6-8).

Figure 3.


