SEEP: Scalable and Elastic Event Processing

Matteo Migliavacca[*], David Eyers[†], Jean Bacon[†], Yiannis Papagiannis[*], Brian Shand[‡], Peter Pietzuch[*]
[*]Imperial College London, [†]University of Cambridge, [‡]CBCU/ECRIC, National Health Service
smartflow@doc.ic.ac.uk

ABSTRACT

Continuous streams of event data are generated in many application domains including financial trading, fraud detection, website analytics and system monitoring. An open challenge in data management is how to analyse and react to large volumes of event data in real-time. As centralised event processing systems reach their computational limits, we need a new class of event processing systems that support deployments at the scale of thousands of machines in a cloud computing setting. In this poster we present *SEEP*, a novel architecture for event processing that can scale to a large number of machines and is elastic in order to adapt dynamically to workload changes.

1. EVENT PROCESSING

Event processing systems can be found in logistics, fraud detection, real-time data analysis and financial services because they process real-time event data with low latency and high throughput. As the volume of available data increases, scalability of these systems is a key concern. The deployment of such systems as cloud services across thousands of virtual machines (VMs) demands *elasticity*: event processing systems should adaptively and smoothly change their resource consumption, depending on their (dynamic) cost and the processing workload. This facilitates maintaining the benefits of high-rate event processing without requiring that infrastructure be provisioned on the basis of maximum workload.

Many application domains have periodic event workloads with varying granularity (e.g. yearly, monthly or hourly). An elastic event processing system can exploit this fact: it requires resources for high-rate operation only for short periods of time. Elasticity is also useful for applications that dynamically tune the quality of event processing. For example, a video surveillance application may employ a range of image processing algorithms with different computational costs. The system may allocate additional VMs when more costly real-time image processing is required. Finally applications may decide to reduce the monetary costs of processing by freeing VMs, which should result in a graceful reduction in processing throughput.

In this poster we describe the design of SEEP, a novel architecture for a scalable and elastic event processing system that can execute across a large number of VMs. The architecture is based on the idea that event processing can be divided into two phases that distribute *event filtering* and *event computation* across VMs, respectively. This approach enables the system to scale both in terms of event rates and the number of event expressions being executed. Both layers are elastic—the system can dynamically allocate and release VMs executing event processing elements.

2. ELASTIC EVENT PROCESSING IN THE CLOUD

We focus on providing event processing at a Platform-as-a-Service (PaaS) level, targeting application areas such as real-time logistics, interactive simulations (including games), electronic marketplaces, fraud detection (including money laundering, credit card and click stream fraud detection), real-time content analysis (including advertisement and electronic surveillance). These applications require processing that has already been shown to be supported by event processing systems [1].

For an elastic event processing system in a cloud data centre, we posit the following requirements:

Expressiveness: detection of event sequences and sequence aggregation. The above applications face high event data rates and the required processing involves the detection of patterns composed from sequences of events, and aggregation of event data. We do not aim to support expensive analytics, for example, through AI techniques (such as classifiers, neural-nets or GAs)—more complex analytics would need to be performed off-line.

Elasticity: scalability according to workload, acquiring and releasing resources as needed. The system should be able to adapt to varying event rates. Rates could either be varied in a controlled way or be due to unforeseeable workload variations. The system should adapt its resource consumption, taking user requirements into account where possible. It should adapt by acquiring additional VMs, or reorganising the event processing computation and event routing, without interruption or significant performance degradation.

Fault tolerance: reliably maintaining throughput and/or latency The system should handle failures of processing hosts running VMs—potentially degrading processing performance

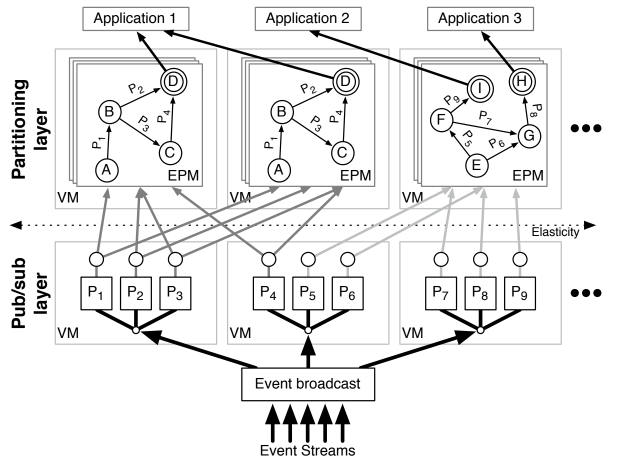

Figure 1: Scalable event processing architecture based on separate layers for event filtering and event processing.

temporarily. Despite this, fault tolerance mechanisms should not significantly impair processing performance in terms of throughput and latency.

To realise an elastic event processing service, we make the following assumptions about the cloud infrastructure. We rely on an Infrastructure-as-a-Service (IaaS) cloud provider such as *Amazon EC2* or *Microsoft Azure* for deployment. We assume that available network throughput and latency, and VM CPU performance make the problem feasible. We also require enough VMs to be available to sustain event processing. Dynamic workload changes are sufficiently predictable to allow timely reaction from our system. In particular, this means that VM migration has to be rare, and that CPU contention on the physical hosts executing VMs does not significantly affect latency.

3. SEEP ARCHITECTURE

SEEP adopts the architecture shown in Figure 1. Event processing is carried out by VMs organised in two layers, a *publish/subscribe* layer and a *partitioning* layer. Conceptually, the publish/subscribe layer exploits the fact that each event processing expression will only require a subset of all events: the layer takes a large number of high-rate event streams and dispatches individual events to VMs in the partitioning layer. The partitioning VMs execute *event processing machines* (EPMs) that implement event processing expressions. EPMs register their interests in events they require for event processing with the publish/subscribe layer.

(a) Publish/subscribe layer. Incoming event streams are broadcast to VMs in this layer using efficient network mechanisms. Each VM maintains a large number of *filter predicates* ($P_1, P_2, \dots P_n$). Filter predicates are applied against the incoming event streams to identify events that need to be dispatched to the partitioning layer. The system creates a distributed index over the predicates to speed up the matching of events [2].

By carrying out filtering first, the system can reduce the volume of events that have to be handled by the partitioning layer. In addition, common interest in the same events by different event processing expressions results in shared predicates and thus more efficient filtering.

(b) Partitioning layer. This layer hosts a large number of EPMs that act as basic building blocks for event processing. Event processing expressions submitted by applications or users are expressed as one or more EPMs. Based on our previous work [4], SEEP adopts an event processing model that uses non-deterministic finite state automata [3, 1]. Each EPM is composed of states ($A, B, \dots$) that represent atomic detection or aggregation steps. States are connected using edges that are associated with predicates ($P_1, P_2, \dots P_n$) on event data and can carry state for stateful event expressions. Events that match predicates trigger state transitions. Matching events are received from the publish/subscribe layer based on these predicates.

This type of automata model is appropriate for a cloud event processing system: (a) due to their simple structure, it is possible to reason about the resource requirements of these automata, and thus to load-balance them across VMs in the partitioning layer; (b) their expressiveness is sufficient for a wide-range of event processing applications since the predicates on edges can include arbitrary computation on events; (c) automata can be split across multiple CPU cores or VMs, which allows the system to handle many concurrent instances of the same automaton, each with its own state.

Scaling processing performance. Both layers in this architecture can exploit the elasticity of a cloud infrastructure by allocating and deallocating VMs in response to changes in the workload: (a) As the rates of incoming events increase, the evaluation of predicates in the publish/subscribe layer becomes more computationally expensive. In response to this, the system can allocate more VMs in this layer and repartition predicate evaluation across the VMs. (b) Similarly, if more event processing expressions are submitted, some of the corresponding EPMs can be allocated to fresh VMs within the partitioning layer in order to balance load.

Even with an unchanged workload, a system administrator may decide to add or remove VMs in order to influence the overall financial cost of the event processing system. Adding more VMs with the same event processing workload increases the processing throughput of the system by spreading out predicate evaluation and EPM execution across a greater number of machines. Similarly, the removal of VMs from the system results in a consolidation that reduces event throughput.

4. REFERENCES

[1] Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker White. Towards expressive publish/subscribe systems. In *EDBT*, volume LNCS 3896, pages 627–644. Springer, 2006.

[2] Françoise Fabret, Hans-Arno Jacobsen, François Llirbat, João Pereira, Kenneth A. Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast publish/subscribe. In *SIGMOD*, 2001.

[3] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Event specification in an active object-oriented database. *SIGMOD Rec.*, 21(2):81–90, 1992.

[4] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter Pietzuch. Distributed complex event processing with query rewriting. In *DEBS '09*, pages 1–12, New York, NY, USA, 2009. ACM.