Towards Transient Resource Usage on Real-Time Stream Processing Systems

Pedro Joaquim
Instituto Superior Técnico, Universidade de Lisboa

ABSTRACT

Stream processing emerged as a key technology, with applications in many different areas. These applications, regardless of their differences, have the common characteristic of continuous processing, thus requiring the system to run on a 24/7 basis. For deployments in public cloud infrastructures, these long term deployments may incur very high costs. However, cloud providers offer transient lived resources, with significantly reduced prices compared to the regular resources, with the proviso that the former can be evicted at any time. Thus, this type of resources is a huge opportunity to significantly reduce operational costs of long lived stream processing systems, missed so far by existing approaches.

In this work we present the initial ideas towards the design of a stream processing system, able to leverage the availability of transient resources on public cloud providers to significantly reduce its operational costs. We identify some challenges of the proposed work, most of them related to the dynamic availability conditions of transient resources, specially problematic for applications that typically have low-latency and high-throughput requirements.

CCS CONCEPTS

• **Computer systems organization** → *Real-time system specification*;

KEYWORDS

Stream Processing, Cloud Computing, Resource Management, Transient Resources

1 INTRODUCTION

Stream processing (SP) [1, 4, 7, 8, 13] recently emerged as a key technology for applications that need to process, in real time, large volumes of information that is being continuously produced. Fraud detection [3] and social trend detection [10], are some relevant examples of such applications. Large companies, like Twitter, Google or LinkedIn, typically deploy SP systems on their own infrastructure [1, 7, 8]. However, the lack of upfront investment on physical infrastructure makes public cloud environments, like Amazon Web Services (AWS) or Google Compute Engine (GCE), appealing solutions for companies without their own infrastructure. Still, because SP is a continuous task that needs to run on a 24/7 basis, even the use of public cloud can accrue to significant costs over the long term.

In parallel, cloud providers have offered for some time the possibility for users to acquire spare resources at a very reduced cost (typically depending on supply/demand conditions) compared to the traditional reserved price. However, these resources can be unilaterally revoked by the provider without any warning for the user. Due to this somewhat unpredictable availability conditions, we further refer to this type of resources as *transient resources*. Recent work [5, 11, 14] studied ways of leveraging these resources to reduce the operational costs of systems operating on cloud environments. Most of these works focus on offline analytical processes, with relaxed timing constraints, that explore the dynamic availability of cheap transient resources to complete the jobs.

Unfortunately, the dynamic availability conditions of transient resources, that can be evicted without notice, may compromise the timeliness guaranties desired for SP applications. Therefore, making them a doubtful choice for systems with real time performance requirements such as the ones that we consider in this work. Nevertheless, the potential cost benefits of a solution that successfully uses transient resources to fully (or even partially) deploy an SP system without compromising its performance requirements, make it a research topic worth investigate.

Furthermore, most public cloud providers allow users to select some of the physical characteristics of the machines (even for transient machines), such as number of CPU cores, memory size, disk type, among others[1]. As different applications running on SP systems may benefit from the use of specific hardware characteristics [8] on particular system components, such as, enhanced network capabilities for tasks that are communication-bound or increased disk throughput for tasks that are I/O-bound. The resource heterogeneity available on public cloud infrastructures further exacerbates the resource provisioning process that needs to consider the hardware characteristics of the machines, their cost and also their availability conditions. This creates a complex trade off between cost and performance that needs to be translated into a suitable strategy to acquire resources based on the system characteristics. Unfortunately, existing approaches do not address the underlaying physical resource provisioning process as they often assume an already existing infrastructure [1, 7, 8].

In short, although a significant amount of research has been dedicated to help automated deployment of cloud applications, and also to systems that aim at leveraging transient resources, existing approaches fail to address the complexity of SP applications that combine the need for heterogeneous hardware, dependability, and timeliness requirements.

2 PREVIOUS WORK

2.1 Stream Processing Systems

Recent systems for stream processing, such as Google's MillWheel [1], Twitter Heron [7] and Apache Samza [8] ease the creation of stream processing applications by decoupling the computation logic from resource management. In this model, users are just required to specify the computational process as a directed graph of computations that receive streams of data and run arbitrary user

[1] https://aws.amazon.com/ec2/instance-types/

code using platform specific hooks to perform standard stream processing operations, such as time windowed tasks and split/merge streams of data. The system is then responsible for deploying the user defined topology into the existing resource infrastructure.

These systems rely on schedulers like YARN [12] and Mesos [6] to assign the computational tasks to existing physical resources. Although these systems release the underlaying SP system from the need to efficiently manage the underlying resources, they are completely oblivious to the resources' acquisition and release management policies. Recently, Dhalion [4] proposed a set of techniques to self-regulate the Heron [7] system for a target performance. Still, the system focuses solely on the high level component management through the scheduling service, therefore also ignoring the underlying physical resources' provisioning policies. As mentioned before, this is specially problematic when considering deployments in public cloud environments, where the user is required to select from a wide range of heterogeneous machines, not only in their hardware characteristics but also in terms of dynamic cost functions and availability conditions.

2.2 Using Transient Resources

Previous research has shown that the judicious use of heterogeneous resources, namely transient resources, can offer significant cost savings. Unfortunately, their solutions cannot be trivially adapted to SP systems as most of them leverage the relaxed timing constraints of offline analytical processes, without real-time and low-latency requirements, to explore the dynamic availability of transient resources to reduce the operational costs.

For example, Pado [14], a data processing engine, tackles the problem of reducing the performance impact of transient resource's revocations. The system receives two inputs: (i) the information about the available machines, which are either reserved or transient and (ii) the computational process translated into a directed graph of computations, similar to the SP systems described before. The system then chooses the task placement in a way that minimizes the performance impact of transient resources revocations by placing the ones that, if evicted, generate long re-computation chains on reserved resources and others on transient servers. Still, Pado assumes the underlying machines to be selected by someone else and it does not ensure a given performance threshold for the system.

Flint [11], a batch-interactive framework based on Spark, tackles the problem of bounding the performance impact of transient resource's revocations on interactive applications by leveraging different spot markets. Given the number and type of machines to use, the system uses historical data of Amazon spot-instances to identify the regions which have the most uncorrelated failures for that target machine type, i.e., the markets where historically a revocation of type M machines do not happen at the same time. Flint then deploys the system on machines across this different markets. Unfortunately, not only does Flint rely on the user to select the number and type of machines to deploy, but also the strategy used to bound the performance impact is not well suited for systems whose computational flow highly depends on machine communication, as machines can be placed at distant geographical locations.

2.3 Automated Deployment

There has been a growing interest in researching for techniques that allow to automate the selection of the most appropriate deployment for a given workload. Systems like CherryPick [2] seek to find the optimal VM configuration for a given workload *provided a cost function* that is assumed to be stable. The dynamic price characteristics of transient resources, together with their somewhat unpredictable failure models, make the integration of these systems in settings that aim at leveraging transient resources a non trivial problem.

3 GOALS

Analysing the existing SP systems, which we briefly described earlier, we identify some key problems that we would like to address in future work. First, most of these systems ignore resource provisioning process by often assuming an already existing underlying physical infrastructure. This is particularly problematic for public cloud environments, where users have to pick the number and types of machines to use. Second, SP systems are long lived and often have 24/7 availability requirements that can accrue to significant costs over the long term. However, existing systems ignore this monetary aspect and do not explore ways of reducing the operational costs of these systems.

In this context, the high level goal of our work is to address the above mentioned problems for public cloud deployments of SP systems. This translates into providing a resource configuration that aims to be the cheapest possible while guaranteeing the performance requirements for the system. We split this high level goal into three, more concrete, goals. Namely:

Goal 1: Leverage the dynamic availability and hardware heterogeneity of transient resources to reduce the operational costs of SP systems in public cloud environments.

Goal 2: Provide performance guarantees, even when using transient resources with dynamic availability conditions.

Goal 3: Use the knowledge about the SP system and its target applications to automatically provision the underlying physical infrastructure in cloud environments, requiring minimal user interaction.

These goals require improving existing SP systems to make them tolerate a possibly large number of evictions efficiently. Furthermore, we need also to design new automated tools to adjust the way the system is provisioned, taking into consideration the underlying application characteristics and in response to dynamic changes in the cost of transient resources.

4 CHALLENGES

We envision a number of challenges that need to be overcome to fulfill our goals. Namely:

- Adapt the fault-tolerance mechanisms of SP systems to minimize the impact of frequent transient resources' evictions. One should consider the trade-off between fault tolerance mechanisms that provide a stable or bounded performance impact in the occurrence of failures, even if adding some

overhead to the steady state performance, and optimistic fault-tolerance mechanisms that have little to no impact during normal execution but take longer to recover.
- Design a resource provisioning strategy that is able to reduce the operational costs of SP systems by leveraging the availability of transient resources and, at the same time, provide minimum performance guarantees.
- Design efficient reconfiguration mechanisms. The system must keep pace with transient market price trends as they have a direct impact in the operational cost of the system. Machines that are very cheap at one moment in time can become very expensive (even above the reserved price). Taking this into consideration, the underlying physical resource (re-)provisioning needs to be done efficiently, as it may be necessary to replace the machines being used.

5 RESEARCH DIRECTIONS

The current line of investigation is still in an early research stage. In the following sections we briefly describe the research directions currently being investigated.

5.1 Component Replication

Systems that have low latency and high throughput requirements are particularly vulnerable to component failures. In the presence of failures these systems may fail to meet a target performance Service Level Objective (SLO). For systems that have such strict performance requirements, one possible solution is to provision active backup instances that ensure the system is able to meet a target performance SLO under a bounded number of component failures. Strategies that leverage recent trending technologies (such as RDMA [9]) are worth looking into to reduce the inherent performance impact.

5.2 Heterogeneity Driven Provisioning Strategies

Knowledge about the performance of specific cluster configurations, possibly with different hardware characteristics, needs to be translated into a suitable strategy to acquire and release resources based on the expected trade-off between cost and performance.

This process should avoid common pitfalls when using transient resources, such as allowing a single eviction that affects multiple machines to leave the system with an insufficient number of resources to meet a target performance SLO, or even completely deprived of resources. For this problem, strategies like the one suggested in [11] should be considered and extended for a component replication setting. One should leverage past market data to infer the correlation between the price trends of different transient machine types (which directly translates into eviction correlation). Using this information one is then able to use transient machines with uncorrelated failures in a replicated setting.

Recent work [15] studied ways of provisioning machines with uncorrelated failures. However, the techniques used are for correlated hardware failures and require privileged information from the provider that is often not available. Therefore, we consider this work orthogonal to the transient resources evictions correlation, that is dependent on market trends and not shared hardware.

5.3 Efficient Resource (Re-)Provisioning

Due to the frequent changes in the underlaying physical resources, either as a result of revocations or planned reconfigurations made to meet target performance or cost goals, the system should handle the frequent changes in resources gracefully and without much impact in performance. When changes are planned, or revocations get preceded by a timely warning (not guaranteed), the system may leverage the underlying component replication to perform phased resource replacements with graceful component shutdown, resulting in no loss or reprocessing of updates [7].

6 EVALUATION SCENARIOS

After identifying a solution that meets the defined goals for this work, the resulting system should be evaluated to validate them. Although it is difficult to predict the exact scenarios in which the final system will be evaluated, as this depends on the solution details that may or may not be ones that we here proposed, some more general aspects can be identified beforehand include:

- The evaluation must include executions in real world scenarios, i.e., execute the system in a public cloud environment. As a possible final solution includes transient resource usage, we consider that the transient behaviour of machines must be simulated through injected faults, following a possible real scenario observed in historical data. This way it is possible to find interesting cases to test and provide a equal setting for possible different implementations that we seek to compare.
- In order to provide long term cost analysis, a simulated execution over historical data of transient resource market prices is required. This historical data allows us to simulate the system behaviour (for example, number of evictions, machines selected for different periods in time) over a period of time that would be otherwise impossible to measure.

7 NEXT STEPS

After having identified the goals of our work, as well as the possible research directions, we now intend to start implementing, testing and adapting the possible solutions identified here. For this early design stage, and also for a possible final working prototype, we intend to use the Heron [7] SP system for a couple of reasons. First, most SP systems follow the same computational model where applications are defined as a directed graph of computations that process a stream of data. This makes our solution valid and easy to adapt for other available SP systems. Second, it is an open source system with a very active community and detailed documentation that eases the development process. Naturally, this choice will need to be reassessed, based on the experience that we will collect from the prototype.

ACKNOWLEDGMENTS

This work was partially supported by national funds through Instituto Superior Técnico, Universidade de Lisboa, and Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

REFERENCES

[1] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle. 2013. MillWheel: Fault-tolerant Stream Processing at Internet Scale. (2013).

[2] O. Alipourfard, H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.

[3] F. Carcillo, A. Pozzolo, Y. Borgne, O. Caelen, Y. Mazzer, and G. Bontempi. 2017. SCARFF: a Scalable Framework for Streaming Credit Card Fraud Detection with Spark. *CoRR* (2017).

[4] A. Floratou, A. Agrawal, B. Graham, S. Rao, and K. Ramasamy. 2017. Dhalion: Self-regulating Stream Processing in Heron. *VLDB* (2017).

[5] A. Harlap, A. Tumanov, A. Chung, G. Ganger, and P. Gibbons. 2017. Proteus: Agile ML Elasticity Through Tiered Reliability in Dynamic Resource Markets *(EuroSys '17)*. Belgrade, Serbia.

[6] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica. 2011. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center *(NSDI '11)*. USENIX, Boston, MA, USA.

[7] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. Patel, K. Ramasamy, and S. Taneja. 2015. Twitter Heron: Stream Processing at Scale *(SIGMOD '15)*. Melbourne, Victoria, Australia.

[8] S. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. Campbell. 2017. Samza: Stateful Scalable Stream Processing at LinkedIn. *VLDB* (2017).

[9] M. Poke and T. Hoefler. 2015. DARE: High-Performance State Machine Replication on RDMA Networks. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '15)*.

[10] L. Recalde, D. Nettleton, R. Baeza-Yates, and L. Boratto. 2017. Detection of Trending Topic Communities: Bridging Content Creators and Distributors. *CoRR* (2017).

[11] P. Sharma, T. Guo, X. He, D. Irwin, and P. Shenoy. 2016. Flint: Batch-interactive Data-intensive Processing on Transient Servers *(EuroSys '16)*. London, UK.

[12] V. Vavilapalli, A. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator *(SoCC '13)*. Santa Clara, California.

[13] S. Venkataraman, A. Panda, K. Ousterhout, M. Armbrust, A. Ghodsi, M. Franklin, B. Recht, and I. Stoica. 2017. Drizzle: Fast and Adaptable Stream Processing at Scale *(SOSP '17)*. Shanghai, China.

[14] Y. Yang, G. Kim, W. Song, Y. Lee, A. Chung, Z. Qian, B. Cho, and B. Chun. 2017. Pado: A Data Processing Engine for Harnessing Transient Resources in Datacenters *(EuroSys '17)*. Belgrade, Serbia.

[15] E. Zhai, R. Chen, D. Wolinsky, and B. Ford. 2014. Heading Off Correlated Failures through Independence-as-a-Service. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association.