Adaptive Application Performance Management for Big Data Stream Processing

Holger Eichelberger, Cui Qin, Klaus Schmid
University of Hildesheim
Universitätsplatz 1
31141 Hildesheim, Germany
+49 5121 883 40334
{eichelberger, qin, schmid}@sse.uni-hildesheim.de

Claudia Niederée
L3S Research Center
Leibniz Universität Hannover, Germany
+49 411 762 17796
niederee@L3S.de

ABSTRACT

Big data applications with their high-volume and dynamically changing data streams impose new challenges to application performance management. Efficient and effective solutions must balance performance versus result precision and cope with dramatic changes in real-time load and needs without over-provisioning resources. Moreover, a developer should not be burdened too much with addressing performance management issues, so he can focus on the functional perspective of the system For addressing these challenges, we present a novel comprehensive approach, which combines software configuration, model-based development, application performance management and runtime adaptation.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Performance attributes, C.1.3 [**Computer Systems Organization**]: Other Architecture Styles – Data-flow architectures, D4.8 [**Software**]: Performance – Modeling and prediction, Monitors

General Terms

Algorithms, Management, Measurement, Performance, Design.

Keywords

Adaptive performance management, resource adaptation, stream processing, algorithm families, Big data, QualiMaster.

1. INTRODUCTION

Risks in financial markets have become a large concern for economies, politics, and the society at large due to their severe consequences. Identifying risks well in advance would have significant economic benefits to regulators and institutional investors. However, analysis of systemic risks [8] is a challenge, as markets and market players are often highly inter-connected. In-depth analyses allowing effective countermeasures require low-latency processing of huge data sets arriving in real time with varying runtime characteristics, e.g., bursty streams if markets become hectic. This situation is aggravated if – for a more in-depth analysis – data from further sources must be analyzed simultaneously, e.g., to identify financial trend indicators from Social Web data.

From a performance point of view, the problem can be framed as follows: There is a need for a data processing pipeline, which at least supports soft real-time constraints, i.e., low latency between data input and risk identification. Computation is performed on a resource pool containing various resources, including specialized hardware, which should be optimally exploited. A processing pipeline can be constrained to the resource pool as scaling out to external resources is not desirable, e.g., for cost reasons or due to data protection rules. Ultimately, the stream characteristics may vary by several orders of magnitude requiring a situation-dependent reconfiguration of the resource usage or actual data analysis.

Current approaches focus on resource-aware configuration / adaptation (e.g., [9]) or runtime performance management (e.g. [10]). In contrast, our approach for the *simplified development of adaptive application performance management mechanisms* smoothly combines resource-aware configuration with model-based code generation and adaptive performance management, in order to ensure optimal reuse of knowledge between those phases.

The work we present here is part of the EU FP7 project QualiMaster[1], which aims to build data analysis algorithms and an adaptive infrastructure supporting fine-grained application performance management, and to demonstrate the approach in terms of a case study for analyzing systemic market risks.

2. APPLICATION BACKGROUND

Our research is strongly inspired by the needs of financial risk analysis. However, the problems we aim to solve are relevant for a much wider range of big data applications and we expect that the resulting solutions will in the future be applied to this broader range of applications.

The application use case assumes that financial risk is visible in the markets themselves (e.g., correlation of stock market data that may lead to serious ripple effects) as well as to some degree in social network data such as Twitter. Part of the problem is to identify risks and forecasting indicators in the input data streams. These streams may easily rise to billions of messages a day (approx. 90 million messages per second for the stock exchanges over the world). While the mere total number of events is high, hectic markets, rumors, or catastrophes may suddenly increase the number of events by several orders of magnitude. In this setting, there are three main tradeoffs that must be taken into account:

[1] http://qualimaster.eu

- **Soft real-time:** The longer the delay between feeding the input data into the processing pipeline and observing the output, the lower the utility of the result. This means in particular, that while the number of events per second may vary widely, the processing latency should not.
- **Resource constraints:** There shall be an optimal allocation of algorithms to resources in particular if processing pipelines are resource-bounded. We cannot just overprovision compute power as we aim at minimizing infrastructure costs and the fluctuation of message throughput may cover several orders of magnitude. This is further complicated by a heterogeneous resource pool consisting of standard server hardware and (often expensive) high-performance co-processors, e.g., Field Programmable Gate Arrays (FPGAs).
- **Result precision:** Various algorithms are available for any analysis task. The algorithms differ in time performance, resource usage and precision of the results they deliver. Typically, algorithms with higher precision require more resources. As a result, different algorithm combinations are optimal in different load situations.

To meet the best tradeoffs at runtime, data processing and performance management must be adaptive. In terms of self-* properties [2], we primarily aim at self-optimization including resource-bounded self-scaling of pipelines within the resource pool. In later stages of the project, we also aim at self-optimization, i.e., to enable the system to learn about itself on a longer time scale and to suggest improvements.

3. THE APPROACH

Typically, several algorithms are available to realize a specific analysis task. These algorithms differ in terms of the tradeoff dimensions introduced in Section 2. We consider algorithms with similar functionality but different runtime tradeoffs as an algorithm family as illustrated for correlation computation in Fig. 1a). We use algorithm families to construct adaptive data analysis pipelines, i.e., data flow graphs consisting of sources, algorithm families (rather than individual algorithms) and sinks. Fig. 1b) illustrates a simple example pipeline using the family from Fig. 1a). Based on this notion, the adaptation is supposed to (1) select the most appropriate algorithms from its respective family at runtime; (2) modify parameters of algorithms, e.g., change filtering / subscription of the input streams, or, to activate data admission; (3) modify the allocation of algorithms to the available resources or adjust the degree of parallelization for a given resource; (4) change the structure of the processing pipeline, e.g., to disable unneeded alternative pipeline paths. Thus, the design of a pipeline also characterizes the adaptation space for runtime performance management.

As the design of the pipelines is essential, we enable the data engineer to configure and validate processing pipelines in terms of a topological product line configuration (introduced in [1]) prior to execution. A configuration consists of the pipelines, algorithm families, algorithms and the available resource pool. Furthermore, our approach[2] allows defining runtime constraints to express Service Level Agreements or explicit resource constraints.

As the configuration is precise enough, we can generate the implementation of the pipelines in a model-based fashion. This ensures that the running system complies with important aspects

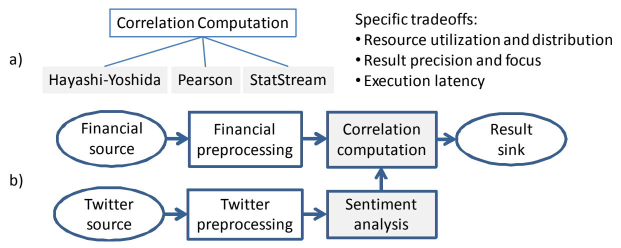

Fig. 1: Example algorithm family a) and pipeline b).

of its model. Further, generating pipelines hides technical details from the data engineer, such as runtime algorithm switching or monitoring probes. Creating such configurations and turning them into code is supported by the toolset EASy-Producer [4] and a graphical pipeline configuration tool on top[2].

At runtime, pipelines are executed by the QualiMaster infrastructure, which realizes the adaptive performance management. On the software side we use the stream processing framework Apache Storm[3], as it is available, stable and widely used in industry. Storm is executed on a cluster of standard server hardware. On the hardware side, we use Maxeler[4] Data Flow Engines (DFEs) and integrate them as (transparent) co-processors. We discuss now our approach to adaptive performance management in terms of the MAPE-K (Monitoring, Analysis, Planning, Execution, Knowledge) model [7], which is realized by corresponding layers of the QualiMaster infrastructure.

Due to the heterogeneous nature of the infrastructure, *Monitoring* aggregates runtime information from various sources. On the software side, we use the application-level runtime statistics of Storm through Apache Thrift[5] (execution time, processing capacity, processed items) as well as SPASS-meter [5]. SPASS-meter is a flexible monitoring framework, which allows observing the resource usage (execution time, memory, network and file transfer) of individual software components. SPASS-meter aggregates raw resource consumptions on various levels, in particular in configurable monitoring groups that can consist of arbitrary methods and classes. This allows us to obtain the resource usage of individual algorithms or components rather than entire JVMs. For code-level application-specific monitoring, we augmented SPASS-meter by a low-overhead probe plug-in mechanism. For high-level monitoring that requires infrastructure interfaces, we generate monitoring probes into the pipeline code. For the hardware side, monitoring utilizes a proprietary network interface to obtain the available DFEs at runtime.

The *Analysis* phase uses the monitoring data to determine deviations from the desired behavior. Currently, we work on a simple scheme that focuses on over- and under-utilization in the form of a water-marking system: If the utilization (capacity) is too high, a reconfiguration changes the parallelization, the resource mapping or the algorithm selection to require fewer resources. If the utilization is too low, a respective reconfiguration leads to a higher utilization and (usually) to a more precise data analysis. In the future, we aim at extending this by constraint violation triggers and a more intelligent, predictive analysis. As the performance tradeoffs of the algorithms are rather different, we aim at algorithm profiles, which reflect the main tradeoffs depending on algorithm parameters, input load and physical

[2] See deliverables D4.1 and D4.2 at http://qualimaster.eu

[3] http://storm.apache.org

[4] http://www.maxeler.com

[5] https://thrift.apache.org

distribution in the cluster (stream processing algorithms are typically distributed). While offline profiles will be captured from a test bench on (ground truth) data with known stream characteristics, online profiles will be derived from monitoring algorithms in a real infrastructure. In addition, we aim at using trend analysis / forecasting as well as prediction of impacting events through Social Web Data to achieve a better reconfiguration. Ultimately, the approach shall support the adaptive execution of several pipelines and analyze their tradeoffs.

In the *Planning* phase, we determine changes to the runtime configuration triggered by the analysis phase, e.g., due to unbalanced resource usage or violated constraints. Here, we combine configuration information with an adaptive planning method. As also domain experts shall be able to adjust the planning, we use as a basis the known concept hierarchy of strategies, tactics, and actions by Stitch [3, 6]: A strategy defines a (high-level) adaptation objective capturing the logical aspect of planning. A tactic describes the technical realization of a potential adaptation using atomic actions. In our approach, actions modify the configuration, cause a runtime validation and, if successful, translate the modifications into infrastructure commands such as changing or migrating an algorithm. It is worth to note that also actions for code generation are available, so that we can optimize algorithm code or even create new pipelines at runtime.

Finally, the *Execution* enacts the changes given as infrastructure commands. An infrastructure command can affect single as well as multiple resources at the same time, e.g., changing an algorithm can cause a migration of the algorithm from software to hardware, and, thus may need specific synchronization. However, the design decisions that we made also limit the reconfiguration capabilities. In particular, the rebalance operation of Storm restarts one (or more) pipelines with new settings, takes even for a trivial pipeline at least 8 seconds and, thus, is not suitable for reconfiguration of real-time streams. To mitigate this issue, we created a customized version of Storm, which allows changing the parallelization of algorithms and migrating their threads or processes at runtime. Our version operates at the same throughput (measured at 1000 items per second) as the original version. As first experiments indicate, it allows us to change the resource allocation of threads in less than 200ms and of processes in two phases of 200ms (separated by a Storm-specific polling time). It is worth mentioning that loading a new algorithm into a DFE can take up to 1 second due to technical restrictions of the FPGAs.

Most of the adaptation *Knowledge* has already been described. This includes the configuration space, the configurations, the observations and performance requirements. The configuration space is characterized by the customizable pipelines, algorithm families, algorithms and the resource pool. The configurations include the initial user configuration, the runtime configuration reflecting the actual state and the desired configuration after execution. Runtime observations are provided by the infrastructure monitoring and include continuous performance, resource usage and precision information. Performance requirements are expressed through constraints and algorithm profiles. To enable self-optimization, we aim at learning this knowledge at runtime, e.g., online capturing of algorithm profiles.

4. CONCLUSIONS

Our approach simplifies the development of adaptive application performance mechanisms by combining resource-aware configuration with model-based generation and adaptive performance management: We model adaptive pipelines in terms of a topological configuration and realize them by model-based code generation. At runtime, the QualiMaster infrastructure monitors the execution, decides about reconfigurations based on the configuration as well as tradeoffs and coordinates the changes.

Initial results are encouraging, but indicate also that adaptive performance management of such an infrastructure is a challenge. We already extended Apache Storm to enable fast resource reallocation at runtime. Recent experiments show that switching among stateless software algorithms can be achieved in less than 50ms, but the reconfiguration time increases with the execution latency as slow algorithms require more input queuing. To optimize this behavior, we are working on a combination of resource reallocation, queuing and, more generally, state transfer among algorithms. Several more experiments will be done in the future, e.g., along the tradeoffs mentioned above or, in particular, determining the impact of adaptations in extreme situations on the data precision. Further, we are working on offline and online analysis of algorithm performance profiles to improve systematic algorithm development as well as adaptive performance management. Moreover, we plan to investigate the limitations of generic performance measurements vs. application-specific (generated) monitoring probes.

5. ACKNOWLEDGEMENTS

This work was partially supported by the European Commission in the 7th framework programme through the QualiMaster project (grant 619525).

6. REFERENCES

[1] T. Berger, S. Stanciulescu, O. Øgård, Ø. Haugen, B. Larsen, and A. Wasowski. To connect or not to connect: experiences from modeling topological variability. In *Software Product Line Conference (SPLC '14)*, pages 330–339, 2014.

[2] A. Berns and S. Ghosh. Dissecting self-* properties. In *International Conference on Self-Adaptive and Self-Organizing Systems (SASO '09)*, pages 10–19, 2009.

[3] S.-W. Cheng, D. Garlan, and B. Schmerl. Stitch: A language for architecture-based self-adaptation. *J. Syst. Softw.*, 85(12):2860–2875, December 2012.

[4] H. Eichelberger, S. El-Sharkawy, C. Kröher, and K. Schmid. EASy-producer: Product line development for variant-rich ecosystems. In *Software Product Line Conference (SPLC '14) Vol. 2*, pages 133–137, 2014.

[5] H. Eichelberger and K. Schmid. Flexible resource monitoring of Java programs. *J. Syst. Softw.*, 93:163 – 186, 2014.

[6] N. Huber, A. van Hoorn, A. Koziolek, F. Brosig, and S. Kounev. Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments. *Serv. Oriented Comput. Appl.*, 8(1):73–89, March 2014.

[7] IBM. An Architectural Blueprint for Autonomic Computing, 2006. www-03.ibm.com/autonomic/pdfs/AC Blueprint White Paper V7.pdf.

[8] P. Mertens and D. Barbian. Beherrschung systemischer Risiken in weltweiten Netzen. *Informatik Spektrum*, (4):283–289, 2015.

[9] M. Rosenmüller, N. Siegmund, S. Apel, and G. Saake. Flexible feature binding in software product lines. *Automated Software Engineering*, 18(2):163–197, 2011.

[10] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Intl. Conf. on Cloud Computing (CLOUD '11)*, pages 500–507, 2011.