Synergy: Quality of Service Support for Distributed Stream Processing Systems

Thomas Repantis Vana Kalogeraki
Department of Computer Science & Engineering
University of California, Riverside
{trep,vana}@cs.ucr.edu

Xiaohui Gu
IBM T.J. Watson Research Center
{xiaohui}@us.ibm.com

http://synergy.cs.ucr.edu/

The Problem: A significant number of emerging on-line data analysis applications require the processing of data that get updated continuously, to generate outputs of interest or to identify meaningful events: i) Analysis of the clicks or textual input generated by the visitors of web sites such as e-commerce stores or search engines, to determine appropriate purchase suggestions or advertisements. ii) Monitoring of network traffic to detect patterns that proclaim attacks or intrusions, to filter out traffic that violates security policies, or to discover trends that can help with network configuration. iii) Customization of multimedia content such as audio or video, provided for entertainment or news coverage, to meet the interests, as well as the hardware and software capabilities of target users. iv) Processing of market data feeds for electronic trading, as well as surveillance of financial trades for fraud detection. v) Monitoring of banking and credit card transactions, or phone records to determine trends or anomalous behavior. vi) Analysis of the data collected by sensors when monitoring wildlife, plants, traffic, or the environment for fire, earthquakes, or intruders.

We refer to this kind of continuously updated data as streams and the applications that process this kind of data in real-time as stream processing applications. Stream processing applications consist of software components that operate on a set of input streams to produce a set of output streams. A component can offer functionality as simple as filtering, or correlation, or as complex as transcoding, or encryption. The combined processing of the data streams by all the different components constitutes the execution of a stream processing application.

Because streaming data arrive in high-volumes and high-rates and at the same time the stream processing applications need to produce output within certain time limits to facilitate on-line data analysis, a single machine is often not able to sustain the load of running an entire stream processing application. Therefore, the stream processing components are hosted by different machines, which communicate over a network. We refer to this type of distributed system as a distributed stream processing system [1]. The dispersion of the stream processing components of an application among multiple machines is further dictated by the need for reliability. Distributing component replicas on independent machines that are geographically apart increases the application's availability.

Providing low-latency, high-throughput execution of distributed stream processing applications with bursty data arrivals entails considerable strain on both communication and processing resources and presents significant challenges to the design of a distributed stream processing system. Important research problems include the distribution of the load among machines [3,8,10], the adaptation to dynamic application requirements and to changing resource availability [2,7], and the decentralization and self-organization of the system [5].

Distributed stream processing applications have Quality of Service (QoS) requirements, expressed in terms such as end-to-end delay, throughput, or miss rate. For example, an alert needs to be raised within a certain time frame after an intrusion, or a trading recommendation needs to be made while processing financial data at certain rates. Adhering to such QoS requirements is crucial for the dependable operation of a distributed stream processing system. The first step towards satisfying the QoS requirements of stream processing application is taking them into account during the application composition. However, as the incoming data rates may increase at run-time, due to external events such as a network attack or a rapid popularity growth of some news event, an application execution may cease to adhere to the requested QoS. Under such dynamically changing conditions, providing application QoS is a challenging task. The problem is complicated further by the large scale and the distributed nature of a DSPS. Accurate centralized decisions are infeasible, due to the fact that the global state of a large-scale DSPS is changing much faster than it can be communicated to a single host.

Our Proposal: We have designed and implemented Synergy [9,11], a distributed stream processing middleware. Synergy is a software running on every machine of a distributed system to offer distributed stream processing applications, under Quality of Service constraints, and while efficiently managing the system's resources. Unlike previously developed distributed stream processing systems [1, 4], Synergy offers sharing-aware component composition. Sharing-aware component composition allows stream processing applications that are composed of individual components to utilize i) previously generated streams and ii) already deployed stream processing components. Synergy's component composition protocol [9] takes the user-requested QoS into account when composing the application component graph. This way, the instantiated application satisfies the QoS requirements defined by the user. Furthermore, Synergy employs hot-spot prediction and alleviation techniques [11], to ensure that the application QoS requirements continue to be met at run-time.

The major research contributions of Synergy can be summarized as follows:

- Synergy employes a decentralized light-weight composition algorithm that can discover streams and components at run-time and check whether any of the existing components or streams can satisfy a new application request. After the qualified candidate components have been identified, components and streams are selected and composed dynamically such that the application resource requirements are met and the workloads at different hosts are balanced.

- Synergy integrates a QoS impact projection mechanism into the distributed component composition algorithm to evaluate the reusability of existing stream processing components according to the applications' QoS constraints. When a component is shared by multiple applications, the QoS of each application that uses the component may be affected due to the increased queueing delays on the processors and the communication links. Synergy's approach is to predict the impact of the additional workload on the QoS of the affected applications and ensure that a component reuse does not cause QoS violations in existing stream applications. Such a projection can facilitate the QoS provision for both current applications and the new application admitted in the system.

- Synergy encompasses a framework built on statistical forecasting methods, to accurately predict QoS violations at run-time and proactively identify application hot-spots. In order to achieve this, our prediction framework binds workload forecasting with execution time forecasting. To accomplish workload forecasting we predict rate fluctuations, exploiting autocorrelation in the rate of each component, and cross-correlation between the rates of different components of a distributed application. To accomplish execution time forecasting we use linear regression, an established statistical method, to accurately model the relationship of the application execution time and the entire workload of a node, while dynamically adapting to workload fluctuations.

- Synergy enables nodes to react to predicted QoS violations and alleviate hot-spots by autonomously migrating the execution of stream processing components using a non-disruptive migration protocol. Candidate selection for migration is based on preserving QoS. We employ prediction again to ensure that migration decisions do not result to QoS violations of other executing applications. To drive migration decisions in a decentralized manner we build a load monitoring architecture on top of a Distributed Hash Table (DHT).

We have implemented a prototype of Synergy and evaluated its performance on the PlanetLab [6] wide-area network testbed using a real network monitoring application [12] operating on traces of real TCP traffic [13]. We have also conducted extensive simulations to compare Synergy's composition algorithm to existing alternative schemes. Our experimental results showed that Synergy achieves much better resource utilization and QoS provision than previously proposed schemes, by judiciously sharing streams and processing components during application composition. Our experimental evaluation has also demonstrated high load prediction accuracy, and substantial benefits in application QoS achieved by migration.

In conclusion, we have designed and implemented Synergy, a completely decentralized distributed stream processing middleware that incorporates mechanisms for sharing-aware component composition [9] and hot-spot prediction and alleviation [11]. The experimental evaluation of our techniques has demonstrated substantial benefits in performance and QoS provision, while achieving good resource utilization. More information on the Synergy middleware can be found at http://synergy.cs.ucr.edu

References

[1] D.J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.H. Hwang, W. Lindner, A.S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the Borealis stream processing engine. In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research, CIDR, Asilomar, CA*, January 2005.

[2] F. Chen, T. Repantis, and V. Kalogeraki. Coordinated media streaming and transcoding in peer-to-peer systems. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium, IPDPS, Denver, CO, USA*, April 2005.

[3] Y. Drougas, T. Repantis, and V. Kalogeraki. Load balancing techniques for distributed stream processing applications in overlay environments. In *Proceedings of the 9th International Symposium on Object and Component-Oriented Real-Time Distributed Computing, ISORC, Gyeongju, Korea*, April 2006.

[4] X. Gu, P.S. Yu, and K. Nahrstedt. Optimal component composition for scalable stream processing. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, ICDCS, Columbus, OH, USA*, June 2005.

[5] V. Kalogeraki, F. Chen, T. Repantis, and D. Zeinalipour-Yazti. Towards self-managing qos-enabled peer-to-peer systems. *Self-Star Properties in Complex Information Systems, Hot Topics in Computer Science, Springer Lecture Notes in Computer Science*, 3460:325–342, May 2005.

[6] PlanetLab Consortium. `http://www.planet-lab.org/`, 2004.

[7] T. Repantis, F. Chen, and V. Kalogeraki. Cooperative media processing and streaming. In *7th Annual Industry Day Poster Session, University of California, Riverside, Second Best Graduate Poster Award*, October 2005.

[8] T. Repantis, Y. Drougas, and V. Kalogeraki. Adaptive resource management in peer-to-peer middleware. In *Proceedings of the 13th International Workshop on Parallel and Distributed Real-Time Systems, WPDRTS, Denver, CO, USA*, April 2005.

[9] T. Repantis, X. Gu, and V. Kalogeraki. Synergy: Sharing-aware component composition for distributed stream processing systems. In *Proceedings of the 7th ACM/IFIP/USENIX International Middleware Conference, MIDDLEWARE, Melbourne, Australia*, November 2006.

[10] T. Repantis and V. Kalogeraki. Alleviating hot-spots in peer-to-peer stream processing environments. In *Proceedings of the 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing, DBISP2P, Vienna, Austria*, September 2007.

[11] T. Repantis and V. Kalogeraki. Hot-spot prediction and alleviation in distributed stream processing applications. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, Anchorage, AL, USA*, June 2008.

[12] Stream Query Repository: Network Traffic Management. `http://infolab.stanford.edu/stream/sqr/netmon.html`, 2002.

[13] The Internet Traffic Archive. `http://ita.ee.lbl.gov/html/contrib/LBL-TCP-3.html`, 1994.