Synergy: A Distributed Stream Processing Middleware

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 unpredictable 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], the adaptation to dynamic application requirements and to changing resource availability [2, 7], and the decentralization and self-organization of the system [5].

Our Proposal: We have designed and implemented Synergy [9], 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.

The distinct characteristics of distributed stream processing applications make sharing-aware component composition particularly challenging. First, stream processing applications often have minimum Quality of Service (QoS) requirements (e.g., end-to-end service delay). In a shared processing environment, the QoS of a stream processing application can be affected by multiple components that are invoked concurrently and asynchronously by many applications. Second, stream processing applications operate autonomously in a highly dynamic environment, with load spikes and unpredictable occurrences of events. Thus, the component composition must be performed quickly, during runtime, and be able to adapt to dynamic stream environments. Third, a distributed stream processing system needs to scale to a large number of streams and components, which makes centralized approaches inappropriate, since the global state of a large-scale system is changing much faster than it can be communicated to a single machine. Hence, a single machine cannot make accurate global decisions.

Despite the aforementioned challenges, there are significant benefits to be gained from a flexible sharing-aware component composition: i) Enhanced QoS provision (e.g., shorter service delay) since existing streams that meet the user's requirements can be furnished immediately, while the time-consuming process of new component deployment is triggered only when none of the existing components can accommodate a new request; and ii) Reduced resource load for the system, by avoiding redundant computations and data transfers. As a result,

the overall system's processing capacity is maximized to meet the scalability requirements of serving many concurrent application requests.

Synergy is implemented on top of a wide-area overlay network and supports both data stream and processing component reuse while ensuring that the application QoS requirements can be met. We focus on the end-to-end execution time QoS metric, consisting of both processing delays at different components and network delays between components. The decision of which components or streams to reuse is made dynamically at run-time taking into account the applications' QoS requirements and the current system resource availability. The major research contributions of Synergy's sharing-aware component composition 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.

We have implemented a prototype of Synergy and evaluated its performance on the PlanetLab [6] wide-area network testbed. 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.

In conclusion, we have designed and implemented Synergy, a completely decentralized distributed stream processing middleware that enables the reuse of existing streams and components. Although we have looked at the problem of sharing-aware component composition, significant challenges remain to be addressed, such as addressing load imbalances and machine failures. Having a software prototype of a distributed stream processing middleware enables us to investigate a multitude of research questions in the area.

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.