Self healing in System-S

Gabriela Jacques-Silva · Jim Challenger · Lou Degenaro · James Giles · Rohit Wagle

Received: 5 April 2008 / Accepted: 24 April 2008 / Published online: 28 May 2008
© Springer Science+Business Media, LLC 2008

Abstract Faults in a cluster are inevitable. The larger the cluster, the more likely the occurrence of some failure in hardware, in software, or by human error. System-S software must detect and self-repair failures while carrying out its prime directive—enabling stream processing program fragments to be distributed and connected to form complex applications. Depending on the type of failure, System-S may be able to continue with little or no disruption to potentially tens of thousands of interdependent and heterogeneous program fragments running across thousands of nodes.

We extend the work we previously presented on the self-healing nature of the job manager component in System-S by presenting how it can handle failures of other system components, applications and network infrastructure. We also evaluate the recoverability of the job management orchestrator component of System-S, considering crash failures with and without error propagation.

Keywords Fault-tolerance · Stream processing systems · Distributed recovery

1 Introduction

Stream processing has recently gained interest as a new way to analyze streaming data such as audio, video, chat, voice-over-IP, and email for applications ranging from monitoring customer service satisfaction to fraud detection in the financial industry. Being able to analyze data as it streams rather than storing and using data mining techniques offers the promise of more timely analysis as well as allowing more data to be processed with fewer resources. In this paper, we evaluate some of the autonomic self-healing capabilities of a stream processing system we are prototyping called System-S [1, 10, 15].

In System-S, the observation has been made that streaming analysis lends itself well to componentization and distribution. In particular, large and complex streaming analysis applications can be broken up into small software building blocks that we call process elements (PEs). For example, a filtering PE may consume a first stream and produce a second filtered stream that is consumed by a correlation PE that correlates the filtered stream with a third stream of data. This composition of applications by connecting PE building blocks together is a powerful feature because it allows for rapid development and introduction of new types of analysis by creating new PEs that are able to produce and consume streams. Even more importantly, this division of applications into building blocks connected by streams provides a natural way to distribute the computation task among a cluster of resources; PEs of different types can be placed on different nodes and the streaming data for PEs connected in the application can be communicated over the network. This distri-
bution is particularly important for scaling in streaming applications because the algorithms involved can be compute intensive.

To better understand the complexity, we describe in more detail the architecture of the System-S runtime.

1.1 System-S overview

System-S is designed to be a highly-scaled distributed system meant to support high volume stream processing. It is composed of a number of loosely-coupled runtime components and extensive tooling including an IDE to support composition of streaming jobs, a sophisticated planner, and visualization tools. The system runtime consists of a number of major components (see Fig. 1):

- A framework upon which the job management, dispatching, and individual node controller components are built. We refer to the framework and the components built upon it as the job management component (JMN). The JMN itself is comprised of the central “orchestrator”, the “resource manager” (RMN), and the “master node controller” (MNC) which will be described in more detail.
- An optimizer (OPT) which determines initial placement of processes and initial resource allocations, and continually adjusts both placement and allocations throughout the life-cycle of a job.
- A “stream processing core” (SPC) which manages the streaming communication system,
- A “dataflow graph manager” (DGM) which manages and tracks the graph of connections among the various streaming processes.

The purpose of the system is to support the scheduling, dispatch, and management of streaming jobs. A streaming job is defined to consist of one or more discrete PEs that use a highly optimized communication layer to exchange streaming data. PEs contain application logic and perform specialized functions such as filtering, annotating, segmenting, or joining of streams.

While streaming, componentization, and distribution all provide many advantages and capabilities for System-S, the