Chapter 11
StreamMine

Christof Fetzer
Dresden University of Technology, Germany

Andrey Brito
Dresden University of Technology, Germany

Robert Fach
Dresden University of Technology, Germany

Zbigniew Jerzak
Dresden University of Technology, Germany

ABSTRACT

StreamMine is a novel distributed event processing system that we are currently developing. It is architected for running on large clusters and uses content-based publish/subscribe middleware for the communication between event processing components. Event processing components need to enforce application-specific ordering constraints, e.g., all events need to be processed in order of some given time stamp. To harness the power of modern multi-core computers, we support the speculative execution of events. Ordering conflicts are detected dynamically and rolled backed using a Software Transactional Memory.

INTRODUCTION

The goal of StreamMine project (http://stream-mine.inf.tu-dresden.de/) is to develop a middleware that supports scalable, near real-time processing of streaming data. As the number of events and data sources increases exponentially, the processing power needed to cope with that information has to be scaled accordingly. For example, for real-time detection of call fraud in a telephone system, one would expect to process from 10,000s to 100,000s events per second. Each event carries a few hundreds of bytes of data and one needs to access several kilobytes of stored data to process an event. Hence, one needs to be able to spread the computation across multiple cores and multiple computers to keep up with this event rate.

The natural choice for scaling up such Event Stream Processing (ESP) applications (Babcock et al., 2002) is to distribute their workload (Cherniak et al., 2003). Such distribution can be performed locally – using multiple processors and process-
StreamMine

ing cores available on a single machine (Welsh et al., 2001). It can also be performed in a distributed environment by using multiple machines connected by a network. Such machines can be either connected by a Local Area Network, forming a cluster (Sterling et al., 1995), or by a Wide Area Network, forming a cloud (Hwang et al., 2007). We architect StreamMine for distributed systems consisting of many-core CPUs. The major contribution of StreamMine is the ability to automatically distribute ESP applications across multiple cores and machines. A StreamMine user is presented with a simple interface that allows for an automatic distribution and parallelization of ESP applications. The parallelization and distribution are transparent to the user and are performed automatically by StreamMine.

An ESP application consists of a set of connected filter components. In order to harness the processing power of multiple cores per machine, StreamMine can automatically parallelize filters. If the operations to be performed only depend on the input events, i.e., they are stateless, one can trivially parallelize the processing by creating multiple instances of the filter component. However, if there are dependencies between the events, such a simple replication is not a valid approach: processing events out of order can result in incorrect state and incorrect outputs. In other words, StreamMine needs to – at least logically – process events in a given order. Parallelization is a non-trivial problem that can result in significant development costs and runtime overheads. In order to facilitate event processing in multicore environments, we propose and investigate the speculative execution of events. Speculative execution allows us to process events in parallel that should normally be processed sequentially, even when they are received out-of-order. We use an underlying Software Transactional Memory (STM) (Herlihy & Moss, 1993; Shavit & Touitou, 1995) infrastructure to optimistically process the events in the context of transactions.

The distribution of components across multiple machines poses another set of challenges. The machines used by StreamMine are commodity PCs connected by an IP network. The combination of commodity components and unreliable communication links can result in messages being arbitrarily delayed or dropped. Moreover, there is a very high probability that one or more PCs will crash during an execution. For example, Google’s MapReduce jobs, which run on a cutting edge computing infrastructure, suffered in March 2006 on average about 6 worker deaths per job. These jobs were using on average 268 machines and the average completion time was 874 seconds (Dean, 2006).

For ESP applications, we need to expect that the type of processing of the events and hence, the dataflow of the events, will change during the lifetime of a system. In other words, we need to be able to support incremental changes to the system. Besides supporting a simple way of composing event processing applications, we need to support the merging of multiple event processing applications to reduce the computing resources that are required.

Taking above factors into consideration when choosing the communication infrastructure for StreamMine, we have decided to use a Content-Based Publish/Subscribe (Eugster et al., 2003) system as our communication middleware. Content-based publish/subscribe (pub/sub) systems provide a decoupled and flexible communication infrastructure, which allows for easy composition of N-to-M communication patterns. Moreover, thanks to its decoupling properties, it naturally supports dynamic systems, and unlike point-to-point communication schemes, it allows for easy system reconfiguration and merging of computations. In order to cope with potentially large amount of events and components in the system, we propose a novel approach towards construction of the content-based publish/subscribe systems. We show how one can use Bloom filters (Bloom, 1970) to abstract away the content