Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Aging-related performance anomalies in the apache storm stream processing system

Massimo Ficco [*], Roberto Pietrantuono, Stefano Russo

Università degli Studi della Campania "Luigi Vanvitelli", Via Roma 29, I-81031 Aversa, Italy
Università degli Studi di Napoli Federico II, Via Claudio 21, 80125 Napoli, Italy

HIGHLIGHTS

- Event stream processing (ESP) has recently emerged as a popular paradigm for implementing high-volume data processing applications.
- Software aging is a phenomenon consisting of the performance degradation, or the increase of the failure rate of a program, which can affects popular stream processing technology as Apache Storm.
- Software rejuvenation is a means to prevent aging-related failures, hence to mitigate the impact of aging in Apache Storm.

ARTICLE INFO

Article history:
Received 2 January 2017
Received in revised form 27 July 2017
Accepted 27 August 2017
Available online xxxx

Keywords:
Software aging
Event stream processing
Apache Storm
Cloud

ABSTRACT

Event stream processing has recently emerged as a popular paradigm for implementing high-volume distributed (near-)real time data processing applications. Several open source systems are today available, supporting the development of such applications, many of which developed with the technologies of the Apache Software Foundation. These so called *stream processors* are long-running complex software systems which may be affected by *software aging*, a well-known phenomenon among operation engineers, consisting of a progressive increase in the failure rate or in performance degradation of a software system over time.

We address the problem of identifying symptoms and sources of software aging in the Apache Storm event stream processing system; this helps to identify proper strategies to prevent or mitigate anomalous behaviors in production environments. To this aim, we present an experimental study investigating aging manifestations in a popular system, namely Apache Storm. Results show that Storm presents anomalous behaviors in long runs, which prevent some topologies from working continuously. These can be attributed to software aging, due to Storm internal resource management mechanisms influenced by the garbage collector and the memory assigned to worker processes. We discuss the aging-related Apache Storm behaviors, and we experiment *rejuvenation* actions, showing that they are actually able to remove them.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

Event stream processing (ESP) has recently emerged as a popular paradigm for implementing high-volume data processing applications. While traditional data processing models use to persist data to databases and then execute queries on the stored data, ESP applications perform complex queries on incoming streams of data to produce timely results in reaction to events observed in the processed data [1].

The development of ESP applications is supported by so called *distributed real time stream processing systems*, i.e., software technologies capable of deploying tasks (which are part of the stream processing application) over a cloud architecture or in general in a distributed execution environment. Stream processing systems find application in many fields, including real time analytics, online machine learning, continuous computation. One powerful such system is Apache Storm [2], a free and open-source platform, able to interoperate with lots of technologies belonging to the Apache ecosystem. Storm is used by many big companies – including Yahoo! and Twitter – for advanced real time distributed computation. Stream processing systems usually run for very long time; therefore, they may be affected by software aging.

[*] Correspondence to: Department of Industrial and Information Engineering, Università degli Studi della Campania Luigi Vanvitelli, Via Roma 29, I-81031 Aversa, Italy.
E-mail address: massimo.ficco@unicampania.it (M. Ficco).

http://dx.doi.org/10.1016/j.future.2017.08.051
0167-739X/© 2017 Elsevier B.V. All rights reserved.

Software aging is a phenomenon consisting of the performance degradation or of the increase of the failure rate of a program as it executes [3]. This is usually due to the accumulation of errors that leads the system-internal environment to a state in which such errors are propagated, causing the so-called aging-related failures [4]. Its common causes are memory leaks, data corruption accrual, unreleased file locks, round-off errors accumulation, unterminated threads, file-space fragmentation. Software aging has been demonstrated to affect many complex long-running systems, such as web servers [5], operating systems [6], and even safety-critical systems [7]. Software aging is usually a consequence of software faults, referred to as aging-related bugs – a class of faults may cause failures only after a long period of execution [4].

Software rejuvenation was proposed as a means to prevent or at least delay aging-related failures, hence to mitigate the impact of aging [8,9]. In its simplest form, rejuvenation involves stopping and subsequently restarting the whole software application or parts of it, in a preventive manner. This allows removing the accrued error conditions, by refreshing software internal state. A number of rejuvenation techniques, at various level of granularity (concerning the entire system or even small parts of it), are now available, including: garbage collection, flushing of kernel system structure, preemptive rollback, re-initialization of data structures, memory defragmentation, micro-reboot, virtual machines-level rejuvenation [3]. Algorithms are also available for the optimal scheduling of rejuvenation, i.e. for the problem of *when* to apply rejuvenation [3].

This paper investigates symptoms and effects of software aging phenomena in the popular stream processing technology Apache Storm. Along with other compatible software, such as Apache Kafka (a distributed publish–subscribe messaging system) and ZooKeeper (a distributed configuration/synchronization system), Storm is widely used to set up ESP infrastructures. The investigation is based on experiments with a workload generator as test application, and measurements are taken to detect aging. The data gathered about memory consumption, throughput and the workload itself are analyzed to discover evidences of software aging afflicting the considered stream processing technology. Besides actually revealing aging phenomena, the experiments allow to spot potential causes of the observed anomalies, attributable to the garbage collection and to memory management. This in turn allows to propose and experiment a software rejuvenation solution.

The rest of the paper is organized as follows. Section 2 provides a description of ESP and of the most popular distributed stream processors. Section 3, describes in more detail the software aging phenomenon and the problems of detection and rejuvenation. Section 4 shows the hardware/software test-bed used for the experiments. Section 5 presents and discusses the experimental results. Finally, conclusions and directions of future work are presented in Section 6.

2. Event stream processors

Event stream processors are software platforms capable of manipulating unbounded streams of data, usually fed through sockets or publish/subscribe data distribution systems. Typically, messages are processed as soon as they arrive; parallel computations are achieved by distributing messages among multiple nodes. Messages in a stream can be collected within a temporal window to provide an output that is a function of more messages. The operation on messages include buffering, join, merge and aggregation. Several platforms are currently available; the most popular open-source ones include Tand Storm, Spark Streaming, Samza and Flink, all developed by the Apache Software Foundation.

According to [2], Storm is: "*a free and open source distributed real-time computation system, which makes it easy to reliably process unbounded streams of data, doing for real-time processing what Hadoop did for batch processing. It can be used with any programming language*". Various stream sources (e.g., queuing and databases technologies) can be plugged into Storm. Thanks to the Trident framework [10], it is also able to perform micro-batching operations, treating messages in a stream as batches gathered within temporal windows.

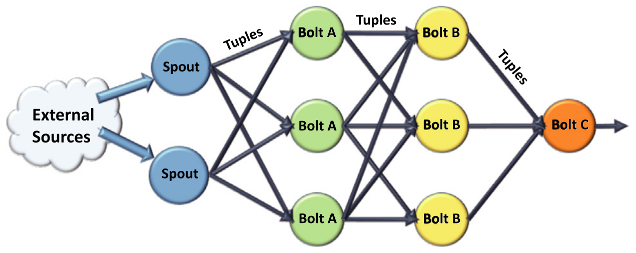

Fig. 1. A Storm topology.

Spark Streaming [11] is not strictly categorized as a stream processor, as it actually performs micro-batch processing, yet it works with unbounded data streams. It does not provide latencies as low as those of Storm, while its performances are comparable to Trident.

As for Samza [12], the main difference with Storm lies in that Samza needs YARN [13]. YARN (Yet Another Resource Negotiator) is a cluster resource manager supporting the separation of the Hadoop Distributed File System (HDFS) from MapReduce, thus granting other system access to HDFS. Samza has a parallelism model which is simpler yet less configurable than Storm. Computation entities in the workflow need to be connected using the Apache Kafka publish/subscribe messaging system (Section 2.3).

Finally, Flink is a general-purpose platform for distributed stream and batch data processing [14], capable of running in standalone mode. It is fully compatible with Hadoop (YARN, HDFS). According to the benchmarking performed by Yahoo!, Flink and Storm show similar performance and latency [15].

2.1. Apache Storm

The architecture of the Apache Storm stream processor [16] is based on the following entities (Fig. 1):

- A *Topology* is a directed acyclic graph, with nodes representing computations and edges data exchanges. Nodes are *spouts* or *bolts*;
- *Tuples* are ordered lists of (untyped) values produced by nodes. Storm needs to know how to serialize values to transfer tuples among nodes;
- *Streams* are unbounded sequences of tuples sent from a node to another. Apart from the very first nodes in a topology, which read from the external data sources, any node can accept more than one stream as input;
- *Spouts* are stream sources; they listen for incoming messages from external sources, and forward them, without performing computation, as their role is solely to emit tuples to the next type of nodes, i.e., the bolts;
- *Bolts* are entities which receive tuples, perform computations, and emit tuples. Tuple transformations include filtering, aggregation, and join;
- *Stream grouping* defines the way (tasks) the tuples are sent among bolts and spouts instances.

Download English Version:

https://daneshyari.com/en/article/6873059

Download Persian Version:

https://daneshyari.com/article/6873059

Daneshyari.com