Automagic Configuration Optimization for Big Data Processing Systems

Muhammad Bilal
Université catholique de Louvain
muhammad.bilal@uclouvain.be

Marco Canini
Université catholique de Louvain
marco.canini@uclouvain.be

1. Introduction

Big data processing systems have dozens of available configuration parameters and system performance crucially depends on their tuning. For example, Apache Storm has about 201 parameters that can be set by the user. Amongst these, there are 49 parameters that can be controlled on a per-topology basis. Users do not need to configure all of these settings; however, a significant number of these parameters will need to be tuned to achieve certain performance goals such as SLOs for mean latency or tail latency and/or throughput for a given production deployment. Moreover, the suitable configurations vary depending upon the available resources, workloads and applications for which the system is being used.

Today, optimizing these configurations is done manually, possibly requiring several hours of investigation and testing by performance engineers. Also, performance engineer need to have considerably detailed knowledge about the systems to figure out a configuration that meets the performance requirements of the deployment. This makes the configuration optimization task a tedious and time consuming one.

The configurations in big data analytics framework deployment can be broadly classified into two categories: resource allocation configurations and application/system specific configurations. Resource allocation has been a subject of significant amount of scientific works, especially for MapReduce jobs [4, 6]. On the other hand, there have been fewer studies in finding out the optimal application-specific configurations for big data frameworks. Most of existing works present solutions specific to MapReduce jobs [2, 3, 5]. Only a few of recent studies have looked into performance optimizations for stream processing systems like Storm [8]; but to the best of our knowledge the broader research challenge remains open.

The goal of this research is to develop a generic framework that can be used to optimize application-specific configurations with the help of developer provided information. The rest of this document is organized as follows: Section 2 discusses preliminary results to motivate the efforts behind this research; Section 3 presents the research directions that we will be investigating; we conclude in Section 4.

Configuration	Possible Values
Number of workers	3, 6, 9, 12
Number of ACKer threads	3 - 12
Number of Spout threads	3 - 64
Number of Split Bolt threads	3 - 64
Number of Count Bolt threads	3 - 64

Table 1. Selected configurations and their values

2. Preliminary analysis

To understand the variation in performance due to system configurations, we conducted preliminary tests on a multi-node Apache Storm cluster. The setup includes 3 nodes, each with a dual CPU with eight cores each and hyper-threading enabled. This means that a total of 32 threads can run on each node, simultaneously. We configure Storm to use the Netty transport layer, as it achieves better performance than with the default, ZeroMQ. Acking is left enabled so as to access performace metrics for our measurements. We use the Rolling word count topology of the Intel Storm benchmark [1], with count window size set to 10s and count emit frequency set to 1s. In Storm, the spout components generate a stream of tuples that are processed by bolt components following an application-specific topology. In this benchmark, each tuple emitted by the spout is a text line which is split into stream of words by the split bolt and the count bolt subsequently counts the occurrences of each words in a time window.

Following an Experimental Design approach, we use a space filling algorithm to cover the configuration space. Table 1 shows the space of possible configurations considered for each experiment. Since our setup has the ability to run 96 threads simultaneously, we have fixed the total number of threads in the topology to 96. We vary the number of threads assigned to each of the topology component while keeping the total number of threads constant. Since the number of possible combinations arising from just these parameters are more than 27k, we used the WSP space filling algorithm [7] to generate 78 configurations for our experiments.

Achieving a suitable trade-off between desired throughput and latency is generally the primary goal of performance optimization in stream processing systems. Hence, we focus on these metrics.

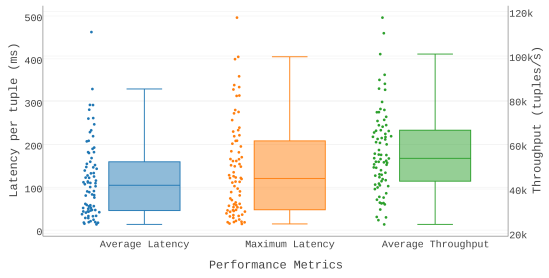

Figure 1. Performance results for selected configurations

Figure 2 shows the variation in different performance metrics. Note that a tuple in the subsequent discussion is a line of text. The per-tuple average latency mainly varies from 13.2ms to 462ms and the maximum latency varies from 14.3ms to 496ms. In addition, there is a significant variation in throughput based on the selected configurations, i.e., from 24k tuples/s to 117k tuples/s. This shows the extent of variation in performance due to distribution of threads among the components of a topology, given a fixed resource budget. Generally, high throughput can be achieved at the expense of lower latency and vice versa. However, we observe that latency and throughput are not always directly correlated. For example, in our test cases, the topology with best average latency has throughput of 42k tuples/s while the topology with worst average latency has 26k tuples/s of throughput.

It is important to note that the reason for obtained latency and throughput for a particular configuration is not always obvious. For example, the CPU and network utilization results of our experiments (not shown here) show that no single resource is being heavily utilized. Thus, it is not the individual resources i.e., CPU and network, that is leading to drastic changes in performance but the interplay of components in the Storm topology. The amount of work done by each topology component and its processing latency, the number of executor for each bolt/spout and their placement, all play a part in the resulting overall performance.

3. Research Directions

Since evolutionary algorithms are suitable for multi-objective optimization problems with complex fitness landscapes, we are looking into leveraging their power to find suitable configurations. The user can define the performance goals that can be transformed into a fitness function for the evaluation of different configurations.

We plan on developing a framework that would allow software developer to provide information about the key configuration parameters, system-specific metrics APIs and performance metrics. We will use this information to automatically improve the application-specific configurations, instead of forcing users to understand the fine details about a distributed software system.

A set of thread-level performance analysis tools might need to be developed to provide the framework with necessary information. An additional direction for this research could be to enable software developers to provide models for their specific software in order to allow for system specific optimizations and potentially use simulations instead of real tests to figure out suitable optimizations, if possible. Since distributed systems are deployed in several settings i.e., bare-metal machines, public Clouds (on VMs) and containers, it will be important to provide support to capture system-level metrics from each deployment setting to get a better picture of resource utilization for optimization.

The outline of the immediate research plan is:

- Investigate the performance issues in stream processing systems to find the factors that lead to degraded performance and quantify their importance.
- Devise set of parameters and building blocks that can be provided to the system developer and used by the framework to assist in configuration optimization of a particular system.
- Investigate/develop analytical and simulation models that can be used for different distributed big data systems.

4. Summary

Given the ever increasing number of big data frameworks and the dozens of configurations that they provide, optimizing each of the systems is becoming a daunting task with significant amount of knowledge required. The goal of this research is to study and reason about the performance issues arising due to configuration parameters. This knowledge would help in development of an automatic configuration selection system that can possibly leverage the information from system developers to apply application-specific optimizations and achieve cost-performance goals.

References

[1] Intel storm benchmark. `https://github.com/intel-hadoop/storm-benchmark`. Accessed: 2016-02-01.

[2] S. Babu. Towards automatic optimization of mapreduce programs. In *SoCC*, pages 137–142. ACM, 2010.

[3] H. Herodotou and S. Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. *VLDB Endowment*, 4(11):1111–1122, 2011.

[4] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *SoCC*, 2011.

[5] K. Kambatla, A. Pathak, and H. Pucha. Towards optimizing hadoop provisioning in the cloud. *HotCloud*, 9:12, 2009.

[6] B. Palanisamy, A. Singh, L. Liu, and B. Jain. Purlieus: locality-aware resource allocation for mapreduce in a cloud. In *SC*, 2011.

[7] J. Santiago, M. Claeys-Bruno, and M. Sergent. Construction of space-filling designs using wsp algorithm for high dimensional spaces. *Chemometrics and Intelligent Laboratory Systems*, 113:26–31, 2012.

[8] M. J. Sax, M. Castellanos, Q. Chen, and M. Hsu. Performance optimization for distributed intra-node-parallel streaming systems. In *ICDEW*, 2013.