How Big Hadoop Clusters Break in the Real World

Ariel Rabkin (student) and Randy Katz
{asrabkin,randy}@cs.berkeley.edu
UC Berkeley/Cloudera, inc.

1 Introduction

Hadoop is among today's most widely deployed "big data" systems. Cloudera is a company offering paid Hadoop services and support. This poster abstract describes lessons from examining a sample of 293 support tickets, from February through July of 2011. We manually labelled the tickets in our sample with the established root cause and the specific system component being worked on. Tickets cover not only the core Hadoop filesystem and MapReduce implementation, but other services, such as HBase, a BigTable clone, and the Zookeeper coordination service.

2 Observations

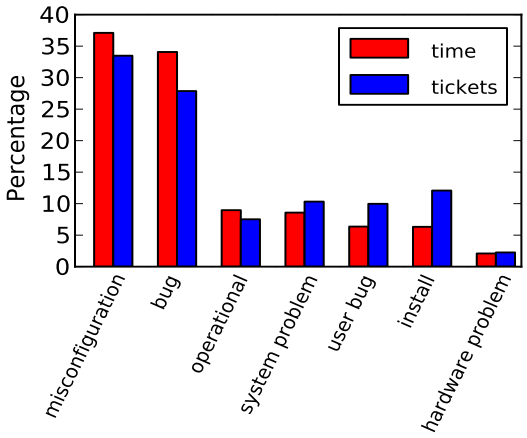

Figure 1: Breakdown of support tickets and support time by issue category. The ratio of "time" to "tickets" is the average time spent resolving an issue.

Figure 1 breaks down the number of tickets and supporter time by root cause. (Supporter time is the estimated time spent by a supporter working on the ticket. For confidentiality reasons, we show only percentages, rather than absolute numbers.) Purely informational or administrative tickets have been excluded. Tickets were classified based on the required action to resolve the issue, after diagnosis. *Bugs* are those that required a patch to the Hadoop platform. *User bugs* required changes to user code. A *system problem* required a change to the underlying operating system in a non-Hadoop-specific way. *Operational issues* are cases where the user is operating the system wrongly, e.g. using the wrong start scripts or shutting down the system abruptly in ways that prevent it from restarting. *Install problems* are those caused by a faulty installation, whether they manifest at installation time or later. *Misconfiguration* includes Hadoop configuration as well as Hadoop-specific permissions and OS-imposed resource limits.

Diagnosis time varies significantly across different types of problems. Install issues tend to be particularly quick to diagnose and fix. They are often deterministic and often manifest at startup, leading to quick debug cycles. Bugs and misconfigurations are both comparatively difficult, since they can require days or weeks of testing to verify if a problem has been fixed. There may also be correlations between a user's level of expertise, the problems they encounter and mistakes they make, and the time required to diagnose an issue.

Misconfiguration represents the largest category of both tickets and of reported supporter time. In Figure 2, we break down the misconfiguration category. There are resource-allocation problems, where some system resource, such as memory, file-handles, or disk-space is being mismanaged or exhausted. There are permissions issues. Malformed or misplaced configuration files are also a significant root cause.

In addition to memory, Hadoop and HBase require the administrator to allocate the number of threads for various purposes, including the sending and receiving sides of the MapReduce shuffle. If there are too many receivers for the number of senders, the receivers will experience frequent timeouts, sometimes leading to jobs aborting. These thread-allocation issues are marked as "threads" in Figure 2.

Most resource misallocation problems are about memory mismanagement. Hadoop has many options for controlling memory allocation and usage, at several levels of granularity. Working from the bottom up, there are various configurable buffers, for things like the MapReduce sort or the Job history maintained in RAM by the Job Tracker. Each Java process itself has a configured maximum heap size. For each daemon, there is an OS-imposed limit on the maximum amount of RAM to be

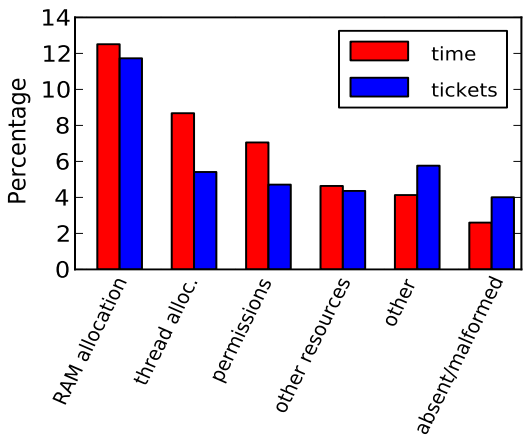

Figure 2: Breakdown of misconfiguration types.

used, the `ulimit`. For MapReduce, the user can tune how many concurrent tasks to execute on each host. And all tasks must fit into physical memory.

Hadoop does not check that these options form a sensible hierarchy. It is possible for the combined MapReduce Task heaps to exceed physical memory, or for the JVM to request more than the OS-imposed limit. Depending whether the JVM heap size, OS limit, or physical memory, this will cause an OutOfMemoryError, JVM exit, or swapping leading to timeouts, respectively.

3 Lessons for System Design

Prune mutually-dependent options Many Hadoop configuration options have well-defined and documented dependencies on one another. This commonly happens with memory allocation, where Hadoop does not check that containment constraints are enforced, and with threading, where the number of senders and receivers can be unbalanced. Closely-coupled options with incompatible settings caused nearly all thread allocation problems and most memory allocation ones as well; this is most misconfigurations and about 20% of all failures.

One way to handle this would be an explicit configuration checker; Cloudera and other vendors have built such tools. This poses some deployment problems, since the checker needs to be kept synchronized with the code and with the best-practices for configuration. A design-time fix would have been to parameterize the configuration space in such a way that one option corresponds to scaling along the best practices dimension, while a second knob allows for workload-specific or "expert" tuning.

Pay attention to memory management Java provides programmers with powerful automatically-growable data structures. This makes it easy to lose track of the size of a given allocation. Problems can arise if a change in workload results in some usually-small data structure becoming too large.

Clearer tracking of credentials Permissions problems arise when users or developers misunderstand what authority a given code region will have. Making permissions more explicit may reduce such errors. Logs should specify what credentials each process has at every point.

Design for [Configuration] change As systems evolve, options are added and removed. Flexible configuration-file formats allow users to set options that appear in a different (past or future) version. Users will be puzzled when setting the option doesn't do anything. Have a strategy in place to look for and flag undefined or obsolete options. (These errors are classified as 'absent' in Figure 2.) Having a narrow XML schema for configuration would help.

4 Related Work

Several previous publications have discussed real-world failure causes. Jim Gray's "Why do computers stop and what can be done about it" is a prominent example, discussing failure data of Tandem computer deployments [1]. Huang *et al* include data about failures seen in production by IBM [2]. Oppenheimer *et al.* looked at failure data from Internet services in the early 2000s [3]. Vishwanath and Nagappan present real-world failure data, although their focus is on hardware, while ours is on software [4].

Our results differ from these previous studies. We found resource and thread allocation to be major issues, while they are barely mentioned in past studies. We suspect that Hadoop deployments are resource-intensive, in ways that Tandem systems were not: Hadoop users are trying to exploit all system resources in a way that Tandem operators were not. The price of this more aggressive usage is new failures.

References

[1] J. Gray. Why do computers stop and what can be done about it. In *Symposium on reliability in distributed software and database systems*, 1986.

[2] H. Huang, R. Jennings, III, Y. Ruan, R. Sahoo, S. Sahu, and A. Shaikh. PDA: a tool for automated problem determination. In *LISA*, 2007.

[3] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do internet services fail, and what can be done about it? In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[4] K. V. Vishwanath and N. Nagappan. Characterizing cloud computing hardware reliability. In *SOCC*, 2010.