Mon May 11, 2009

Hadoop Sorts a Petabyte in 16.25 Hours and a Terabyte in 62 Seconds

by Ajay Anand

We used Apache Hadoop to compete in Jim Gray's Sort benchmark. Jim's Gray's sort benchmarks consists of a set of many related benchmarks, each with their own rules. All of the sort benchmarks measure the time to sort different numbers of 100 byte records. The first 10 bytes of each record is the key and the rest is the value. The **minute sort** must finish end to end in less than a minute. The **Gray sort** must sort more than 100 terabytes and must run for at least an hour. The best times we observed were:

Bytes	Nodes	Maps	Reduces	Replication	Time
500,000,000,000	1406	8000	2600	1	59 seconds
1,000,000,000,000	1460	8000	2700	1	62 seconds
100,000,000,000,000	3452	190,000	10,000	2	173 minutes
1,000,000,000,000,000	3658	80,000	20,000	2	975 minutes

Within the rules for the 2009 Gray sort, our 500 GB sort set a new record for the minute sort and the 100 TB sort set a new record of 0.578 TB/minute. The 1 PB sort ran after the 2009 deadline, but improves the speed to 1.03 TB/minute. The 62 second terabyte sort would have set a new record, but the terabyte benchmark that we won last year has been retired. (Clearly the minute sort and terabyte sort are rapidly converging, and thus it is not a loss.) One piece of trivia is that only the petabyte dataset had any duplicate keys (40 of them).

We ran our benchmarks on Yahoo's Hammer cluster. Hammer's hardware is very similar to the hardware that we used in last year's terabyte sort. The hardware and operating system details are:

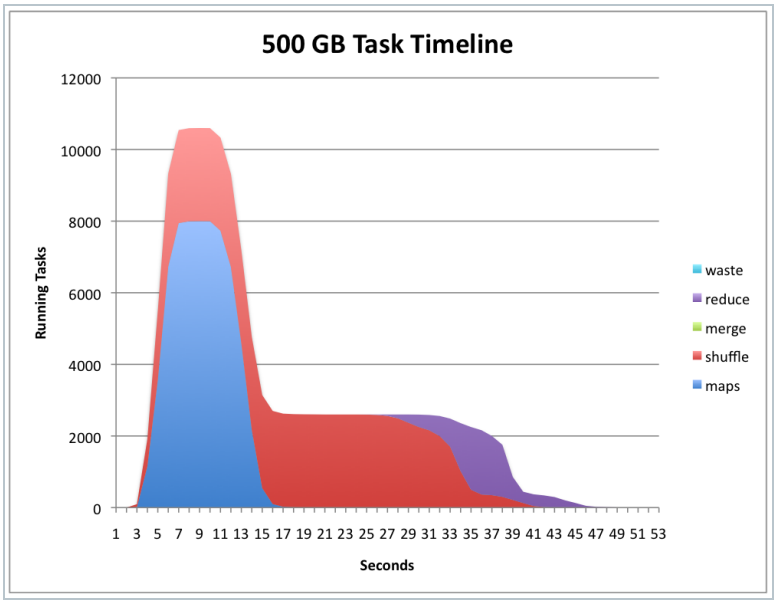
- approximately 3800 nodes (in such a large cluster, nodes are always down)
- 2 quad core Xeons @ 2.5ghz per node
- 4 SATA disks per node
- 8G RAM per node (upgraded to 16GB before the petabyte sort)
- 1 gigabit ethernet on each node
- 40 nodes per rack
- 8 gigabit ethernet uplinks from each rack to the core
- Red Hat Enterprise Linux Server Release 5.1 (kernel 2.6.18)
- Sun Java JDK (1.6.0_05-b13 and 1.6.0_13-b03) (32 and 64 bit)

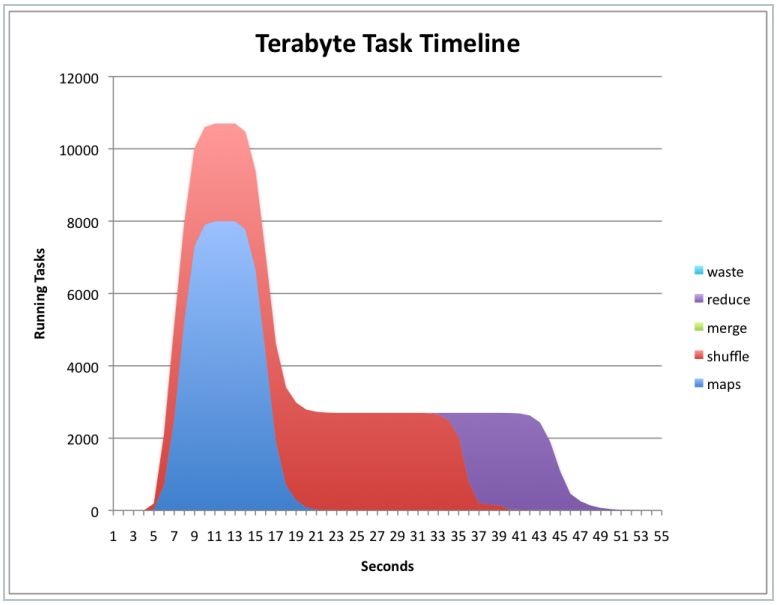
We hit a JVM bug that caused a core dump in 1.6.0_05-b13 on the larger sorts (100TB and 1PB) and switched over to the later JVM, which resolved the issue. For the larger sorts, we used 64 bit JVMs for the Name Node and Job Tracker.

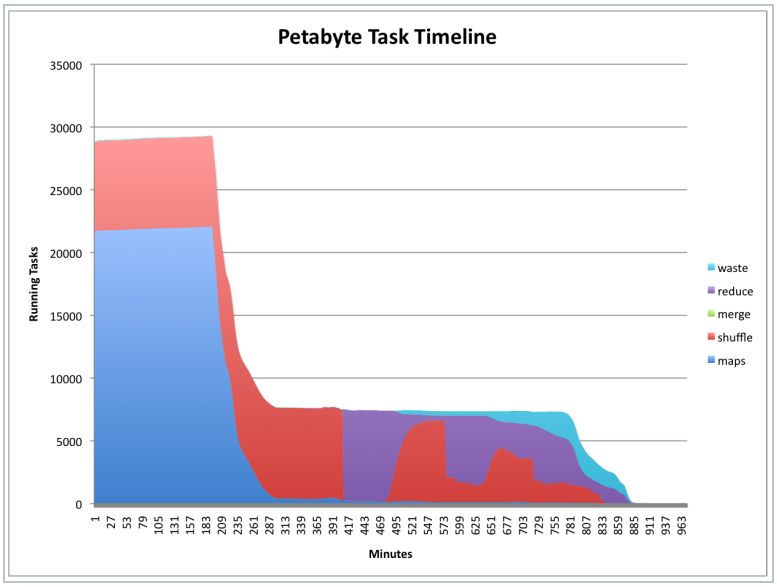
Because the smaller sorts needed lower latency and faster network, we only used part of the cluster for those runs. In particular, instead of our normal 5:1 over subscription between racks, we limited it to 16 nodes in each rack for a 2:1 over subscription. The smaller runs can also use output replication of 1, because they only take minutes to run and run on smaller clusters, the likelihood of a node failing is fairly low. On the larger runs, failure is expected and thus replication of 2 is required. HDFS protects against data loss during rack failure by writing the second replica on a different rack and thus writing the second replica is relatively slow.

Below are the timelines for the jobs counting from the job submission at the Job Tracker. The diagrams show the number of tasks running at each point in time. While maps only have a single phase, the reduces have three: **shuffle**, **merge**, and **reduce**. The shuffle is the transfer of the data from the maps. Merge doesn't happen in these benchmarks, because none of the reduces

need multiple levels of merges. Finally, the reduce phase is where the final merge and writing to HDFS happens. I've also included a category named **waste** that represents task attempts that were running, but ended up either failing, or being killed (often as speculatively executed task attempts).







If you compare this years charts to last year's, you'll notice that tasks are launching much faster now. Last year we only launched one task per heartbeat, so it took 40 seconds to get all of the tasks launched. Now, Hadoop will fill up a Task Tracker in a single heartbeat. Reducing that job launch overhead is very important for getting runs under a minute.

As with last year, we ran with significantly larger tasks than the defaults for Hadoop. Even with the new more aggressive shuffle, minimizing the number of transfers (maps * reduces) is very important to the performance of the job. Notice that in the petabyte sort, each map is processing 15 GB instead of the default 128 MB and each reduce is handling 50 GB. When we ran the petabyte with more typical values 1.5 GB / map, it took 40 hours to finish. Therefore, to increase throughput, it makes sense to consider increasing the default block size, which translates into the default map size, to at least up to 1 GB.

We used a branch of trunk with some modifications that will be pushed back into trunk. The primary ones are that we reimplemented shuffle to re-use connections, and we reduced latencies and made timeouts configurable. More details including the changes we made to Hadoop are available in our report on the results.

-- Owen O'Malley and Arun Murthy

AjayAnand

Categories Announcements

Previous Post « **» Next Post**

MORE FROM AJAY ANAND

Hadoop 0.20.S Virtual Machine Appliance
Hadoop Summit 2010 – Agenda is available!
Hadoop computes the 10^15+1st bit of π
Hadoop Summit 2009 – Open for registration
Using ZooKeeper to tame system test for large-scale services

17 COMMENTS

Ben 104 weeks ago | Report Abuse

Interesting stuff. How many times did you run the operation before finding the optimal amount of nodes and map functions? It is interesting to see such a massive difference in the number of maps for the sort of 100,000,000,000,000 bytes compared to the others.

Permalink

Owen O'Malley 104 weeks ago | Report Abuse

We both experimented with the configuration and made changes to Hadoop that we are starting to roll back into trunk. The 100 TB run we used a relatively large map size of 0.5 GB / map, which is twice what we typically use. With the 1 PB run, we realized it would run faster with even bigger maps, so ran with 12.5 GB / map, which is much much larger than we default to.

In terms of numbers of nodes, we just used two sizes. The “small” was optimized to balance out the number of nodes per an uplink and increase our all to all performance. The “large” was everything. The variation other than that is just caused by which machines were down that day. (Because Hadoop deals with node failures, there is no need to restore the nodes immediately. So operations waits until 10% of a cluster are down and then fixes them all.)

Permalink

Ben 104 weeks ago | Report Abuse

Thank you for the confirmation on that. It is an interesting finding. Its hard to find any sort of information like the above for any large hadoop installation, since so few are publicised.

Permalink

Owen O'Malley 103 weeks ago | Report Abuse

As a follow up, we just sorted 100 TB in 97 minutes, which leads to the same 1.03 TB/minute that we got on the petasort.

Permalink

walter 103 weeks ago | Report Abuse

do “sorting time” include the time to load the 100bytes from disk ? if yes can you compare the time spend “loading” with “working” ?

Permalink

Owen O'Malley 103 weeks ago | Report Abuse

Yes, the overall time includes launching the program, reading from disk, sorting it, and writing it back. To checksum 100 TB of data takes about 10 minutes with 98% of the reading being done in 8 minutes or so.

Permalink

Bradley 103 weeks ago | Report Abuse

The links to jira in the report link to http://issues.apache.org/jira/brows/HADOOP-XXX – the ‘browse’ is missing the e so they don’t work

Permalink

Francisco 103 weeks ago | Report Abuse

Have any simmilar tests been done with hbase? Or is the sort such a simple operation that it was easier to just do it in hadoop?

Permalink

Yavuz 103 weeks ago | Report Abuse

i think if you upgrade the linux kernel you can take better results, please look at the link at kernel-perf.sourceforge.net

http://kernel-perf.sourceforge.net/results.machine_id=35&options=.html

java performance is better with new kernels.

Permalink

tim hawkins 100 weeks ago | Report Abuse

is it posiblen to get a breakdown of the exact components used , ie hadoop version numbers etc, we have been having some stability issues with large datasets, so i would like to cross check against your config.

Permalink

Pablo Navarro Castillo 96 weeks ago | Report Abuse

The report is not accesible, please check the url. Thanks,

Permalink

Rares Vernica 94 weeks ago | Report Abuse

Sorry to be a little off topic. Can you please tell me how did you generate the figures? Thanks!

Permalink

Nasser 90 weeks ago | Report Abuse

BTW, how do you use and connect those 4 SATA disks? RAID? 5,10?

Permalink

Nasser 90 weeks ago | Report Abuse

so sorry for publication. in fact sending four times. It just showed me error and nothing more.

Permalink

ak 79 weeks ago | Report Abuse

Pablo, now these nice graphs can be generated with Hadoop Timelines

Permalink

Aastha 56 weeks ago | Report Abuse

Hi,

I there a way, by which we can separate out the time actually taken by different activities? Like: 1. The time taken for reading the data from hdfs 2. The time taken in processing the data (like sorting of data). 3. The time taken for dumping/ writing back the data/ output to hdfs.

Please let me know if it is yes, and how we can do that?

Regards, Aastha

Permalink

vicaya 50 weeks ago | Report Abuse

Very interesting results!

1PB sort in 16 hours and 15 minutes on 3600 nodes. In comparison, Google sorted 1PB in 6 hours and 2 minutes on 4000 nodes using their proprietary stack 6 months ago:

http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html

Note, the Google stack (GFS/MapReduce) is mostly written in C++, while Hadoop is in Java. OTOH, Google probably have spent 10x more developers' man years in their system than Hadoop though.

Permalink

Post a Comment

You must be logged in to Yahoo! to comment. Log In.