Hadoop EKG: Using Heartbeats to Propagate Resource Utilization Data

Trevor G. Reid
Duke University
tgr3@duke.edu

Jian Wei Gan
Duke University
jg76@duke.edu

***Abstract*—Hadoop EKG is a modification to the Hadoop codebase that allows for the real-time transmission of resource statistics of running nodes to the JobTracker. These statistics include information such as CPU usage, disk IO bandwidth, and available memory of the system. In this paper, we evaluate our design choices in piggybacking resource utilization data on heartbeats, and explain how this data can be useful for real-time task scheduling decisions.**

Our evaluation of the different options–metrics, counters and heartbeats–show that heartbeats are the most suitable for propagation of real-time resource utilization data.

I. Introduction

Hadoop EKG is a modification to the Hadoop codebase that allows for the real-time transmission of resource statistics of running nodes to the JobTracker. These statistics include information such as CPU usage, disk IO bandwidth, and available memory of the system. Hadoop EKG was initially developed due to our desire to develop a more useful task tracker interface that could display more up-to-date and valuable information to a Hadoop user about their map-reduce job. However, thanks to the guidance of Professor Babu and the help of his graduate students our focus shifted from how to display more useful information to what that useful information is and to whom it would be useful. Although HadookEKG was initially developed with the Hadoop user in mind, as our project took shape our focus shifted towards Hadoop itself, particularly in the scheduler. The final iteration of Hadoop EKG was designed in that hopes that a more efficient task-scheduling algorithm could be developed that factors in current resource utilization of nodes to intelligently assign tasks.

The guiding principle when designing Hadoop EKG was to create an integrated, lightweight means of propagating resource usage statistics from TaskTrackers to the JobTracker. Since a channel for transmitting information between these two entities already exists in the heartbeat, we decided to piggyback on this protocol when transmitting the resource statistics. This allowed for a fairly simple implementation that conformed to Hadoops existing design and required no serious changes to the overall code infrastructure. Since we were primarily altering the information sent in heartbeat, most of code modifications were to the TaskTracker and related classes. Also, a plug-in was created in order to actually obtain the resource usage statistics from the operating system.

II. Design Goals and Implementation Options

A. What to Propagate

We wanted to propagate data from the TaskTracker to the JobTracker that would be useful for the JobTracker to schedule tasks. Another question we wanted to answer was what data makes sense to propagating close to real-time. We decided on propagating JobTracker resource utilization because such data would be useful for tagging tasks with what resource they consume a lot of. E.g. you can tag a task as CPU intensive if when running the task, the TaskTracker has CPU usage above a certain threshold. If we can tag tasks with what resources they are using a lot of, we can make real-time scheduling decisions based on this. E.g. you should not schedule many CPU intensive tasks together.

We decided to propagate 3 main types of resource utilization data: CPU usage, memory usage (both virtual and physical) and disk I/O. We decided on these 3 pieces of information because they are the main ways a Task can choke up resources on a TaskTracker.

B. How to Propagate

Our main design goals for Hadoop EKG were to build a system that could propagate information from the TaskTrackers to the JobTracker in real time, and for that system to be lightweight.

First, we had to decide what mechanism we were going to use to propagate information from the TaskTrackers to the JobTracker. The 3 systems that already existed in Hadoop were 1. Metrics, 2. Counters and 3. Heartbeats. Metrics can be run as an external process, while Counters and heartbeats are propagated within Hadoops MapReduce code, and use a Java Remote Procedure Call (RPC.) We wanted to leverage an existing mechanism to propagate data so it would require minimal modification to the Hadoop code and fit into the current design.

1) Metrics: Hadoops NameNode, SecondaryNameNode, DataNode, JobTracker, and TaskTracker daemons all expose runtime metrics. [2] However, a more thorough way to use metrics is to use the Ganglia Monitoring System with Hadoop, which runs as an external process(es) of the Ganglia Monitoring Daemon and Ganglia Metadata Daemon. We decided not to use Metrics as our propagation mechanism because it was not a lightweight solution. Ganglia offers real-time monitoring

metrics, but we were looking for a solution for Hadoop users (as opposed to system administrators) and did not want to require installing heavyweight monitoring tools. Another issue with using metrics was to get the data from the monitoring daemons back to the JobTracker daemon, which required an inter process communication set up, which would have made the system even more complex.

2) Counters: Hadoop maintains some built-in counters for every job, which report various metrics for your job. For example, there are counters for the number of bytes and records processed, which allows you to confirm that the expected amount of input was consumed and the expected amount of output was produced. [**?**] Counters are propagated through RPC and Counters are kept track of within the TaskTracker daemon. There is also an API to add custom Counters, which makes it a great way to propagate data in a lightweight way. However, theyre not sent as often as heartbeats because they have are larger and more bandwidth-heavy. The default counter interval is 1 minute, while the minimum heartbeat interval is 3 seconds. This means that theyre less real-time than heartbeats.

3) Heartbeats: Hadoop has 2 types of heartbeats, one for MapReduce and one for HDFS. We looked at the MapReduce heartbeat since we are trying to propagate data from TaskTrackers to the JobTracker. Heartbeats are used to propagate the status of TaskTrackers. Heartbeats are sent every 3 seconds minimum from the TaskTracker to the JobTracker through RPC. They are sent less often for larger clusters so the JobTracker does not have to process too many heartbeats, and since there is a bandwidth bottleneck at the single JobTracker.

We found heartbeats to be the best mechanism for propagating data because they are lightweight, sent in very short intervals and already have a ResourceStatus that is propagated with the heartbeat that is suitable to add more resource data to.

After deciding on using heartbeats, we had to take into account the amount of data propagated in each heartbeat because the heartbeats are sent in a very short time interval, hence if we propagate too much data with each heartbeat, it will result in heavy bandwidth usage.

III. Implementation

In this section, we go into detail on how our system was implemented and on exactly what code we modified. Our implmentation consisted of 1. getting a thorough understanding of how the system works, 2. porting 0.22 code to work with 0.20.2, and 3. implementing sampling and propagation of more resource utilization data.

A. Versions

At first we tried to used 0.22-alpha, but the build was unstable and we had major issues building and running it, so we decided to go with 0.20.2, since that was what we used for our programming assignments, and it was stable enough for us to build and run. A catch here is that in version 0.22, Hadoop already propagates more resource utilization data from JobTrackers to TaskTrackers through the heartbeat.

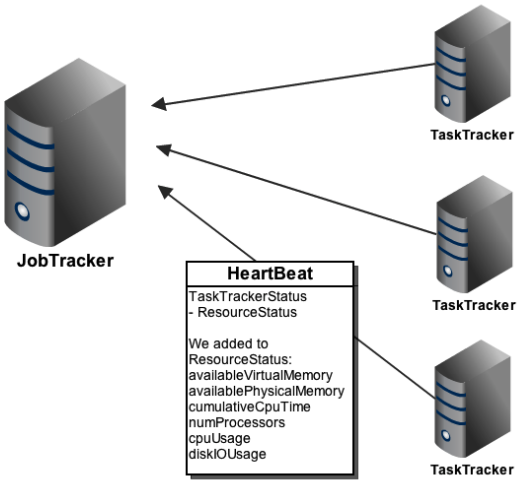

Fig. 1. Hadoop EKG System

B. Heartbeat

In our implementation of using the heartbeat to propagate data from TaskTrackers to the JobTracker, we wanted to fit into Hadoops current design as much as possible.

Our system is shown in Figure 1. At every heartbeatInterval milliseconds, the TaskTracker daemon will prepare a heartbeat and send it to the JobTracker. This heartbeat consists of the following data: a TaskTrackerStatus instance inside this instance, there is a ResourceStatus instance booleans for justStarted, justInited and askForNewTask a short for last heartbeatResponseId

The TaskTracker will only update the ResourceStatus data if askForNewTask is true, i.e. when there are empty map or reduce slots.

We left this implementation, but we recognize there are arguments for always updating the ResourceStatus before each heartbeat instead of only when there are empty slots. If we update it on every heartbeat, the JobTracker and scheduler would be able to tag every task with what kind of resource it uses intensively. However, it incurs the cost of collecting all the resource utilization data with every heartbeat, even if the JobTracker cannot schedule any more tasks on that TaskTracker node.

What follows is a deeper dive into exactly what code we changed in the respective files:

1) /src/mapred/org/apache/mapred/TaskTracker.java: In the method transmitHeartBeat(), the TaskTracker prepares the heartbeat and sends it. If there are empty mapper or reducer slots, it updates the ResourceStatus instance to be sent with the TaskTrackerStatus. We modified code after line 1216 to obtain more resource utilization data and update the ResourceStatus with it. After which, the heartbeat is transmitted to the JobTracker through an RPC–jobClient.heartbeat().

We also added functions that obtain these resource status from the resourceCalculatorPlugin, which defaults to null if

there is no matching resource calculator plugin available. (We only have a LinuxResourceCalculatorPlugin.)

2) /src/mapred/org/apache/mapred/TaskTrackerStatus.java: In TaskTrackerStatus, we modified the static inner class ResourceStatus at around line 58 to contain more instance variables. Before modification, ResourceStatus was already used as a class representing a collection of resources on that TaskTracker, and has fields like totalVirtualMemory. We added more fields such as cumulativeCpuTime and diskIOUsage, along with getters and setters to this class.

Also, this class implements the Writable interface so that it can be written to a buffer to be sent through the network. Hence, to be able to read and write these fields, we needed to overwrite the write() and read() methods of the ResourceStatus class to read and write the extra fields we added. We do this at around line 314.

3) /src/mapred/org/apache/mapred/JobTracker.java: We did not need to modify any code in JobTracker. JobTracker keeps a HashMap of TaskTracker Name to last TaskTrackerStatus received from that TaskTracker (line 1485). The ResourceStatus instances can be accessed through taskTrackerStatus.getResourceStatus().

4) /src/webapps/job/machines.jsp: We modified this page to display the real-time resource utilization on TaskTrackers that we propagated with the heartbeat. A screenshot of this page is shown in Figure 2. This display in the JobTracker is less useful for monitoring, but was more for a proof of concept that we could obtain all the Resources and propagate them through the heartbeat. We added the method generateTaskTrackerResourceTable() on line 81 that displays a table of various resource utilization info and what tasks are being run on each TaskTracker, to show that these Tasks can be tagged as resource intensive after they go above a certain resource utilization threshold, e.g. CPU Usage $> 60\%$.

C. Sampling

All data sampled was obtained from the */proc/* directory in the Linux filesystem. Important files used were: cpuinfo for obtaining statistics such as number of processors and their speed, *diskstats* for IO information, and *stat* for CPU usage information. The files mentioned are all in the */org/apache/mapred/util/util/* package.

1) Disk IO Bandwidth: This calculation is performed in *LinuxResourceCalculatorExtendedPlugin.java*.

The information required for this calculation is obtained from /proc/diskstats which conforms to the format shown in Table I. The values in diskstats are from bootup onwards. In order to get an idea of the number of IOs completed the number of reads issued and number of writes completed were sampled. To obtain a value for IO rate, the number of IOs completed and the CPU time at which this information was sampled were maintained over the period between heartbeat. Then, a rate of IO was calculated using equation (1).

$$\begin{aligned} R_i &= \text{Current Reads Issued} \\ W_i &= \text{Current Writes Completed} \\ R_{i-1} &= \text{Old Reads Issued} \\ W_{i-1} &= \text{Old Writes Completed} \\ CPU_i &= \text{Current Cpu Time} \\ CPU_{i-1} &= \text{Old Cpu Time} \\ \text{IO Bandwidth} &= \frac{(R_i + W_i) - (R_{i-1} + W_{i-1})}{CPU_i - CPU_{i-1}} \end{aligned} \quad (1)$$

2) CPU Usage: This calculation is performed in *LinuxResourceCalculatorPlugin.java*

CPU usage was calculated using the data in */proc/stat* which conforms to the format shown in Table II. In order to get an idea of CPU usage, the values for time spent on normal, niced, and system processes were summed together to obtain the total amount of time the CPU spent executing processes. In a similar manner to the way IO bandwidth was calculated, this information is maintained over the period of heartbeat and percent usage is calculated.

3) Available Memory: This calculation is performed in *LinuxResourceCalculatorPlugin.java*

The information necessary of this calculation is in */proc/meminfo*. This was by far the easiest information to sample because the data simply needed to be parsed from the file and did not require any previous sample information.

IV. Evaluation

A. Lightweight and Extensible

The advantage of using the heartbeat to propagate data from TaskTrackers to JobTracker is that we leverage an existing mechanism that is very well suited for propagating real-time data. More importantly, by extending ResourceStatus to include more resource utilization data, our implementation fits well into the current Hadoop design. The result is a very lightweight and easily extensible system for propagating resource utilization data from TaskTrackers to JobTrackers.

B. Bandwidth Utilization

Our implementation sends 84 bytes of resource utilization data in the ResourceStatus of each heartbeat. A disadvantage of our system is that there more data we propagate in each heartbeat, the heavier the toll we put on bandwidth. This is especially the case since for large clusters, there are many TaskTrackers and only 1 JobTracker, which has to receive heartbeats from all of the TaskTrackers. If the heartbeat were too large, we would clog up bandwidth to that individual JobTracker. Hence there is an upper bound on the amount of data we can propagate through this mechanism.

C. Real-time

We also recognize that the data is only real-time up to the period of the heartbeat interval, which is a minimum of 3 seconds, and larger for larger clusters. However, this is close enough to real-time for purposes of the scheduler tagging tasks as CPU/Disk I/O/Memory intensive.

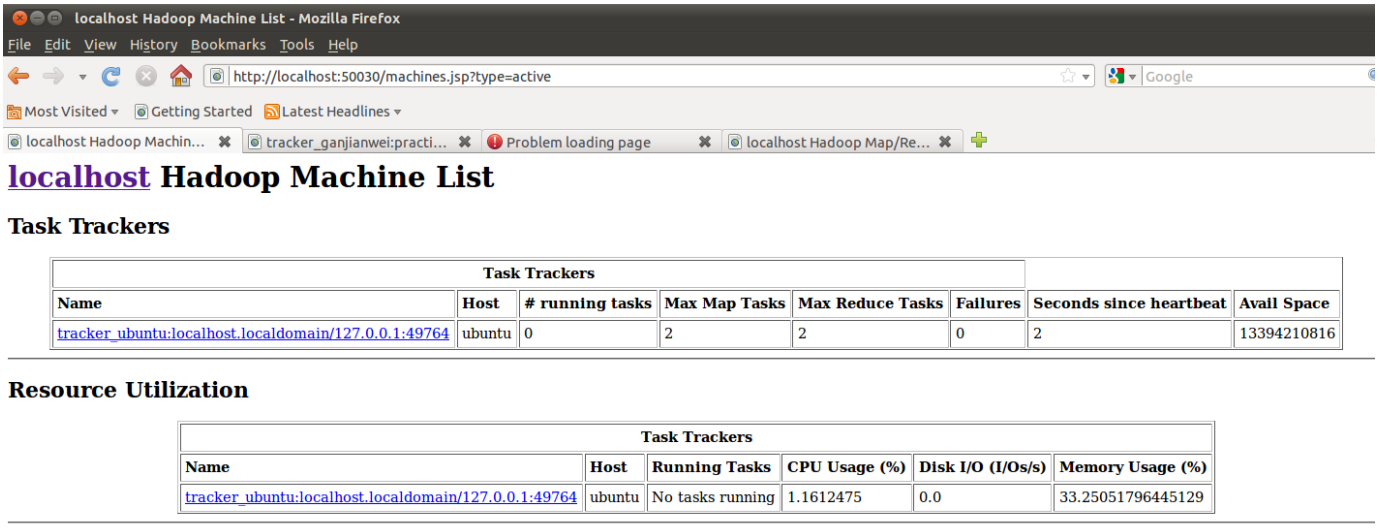

Fig. 2. JobTracker Machines Webpage

D. Synchronous sampling

Our implementation does sampling synchronously in the TaskTrackers transmitHeartbeat() function while preparing the TaskTrackerStatus ResourceStatus before sending the heartbeat. Our sampling is not expensive to the extent where we do sampling over a period of time synchronously in the LinuxResourceCalculatorPlugin, but we need to read from */proc/*, which might require disk I/O, which can be expensive if we do it multiple times, and before each heartbeat where we have empty map or reduce slots.

V. Future Work

A. Smart Scheduling

In its current implementation, Hadoop EKG only obtains and transmits resource information. In no way does the scheduler actually utilize this information. In order to help facilitate this modification to the scheduler we propose to develop a means of tagging tasks as bandwidth intensive for a particular resource. For example, a task with consistently high CPU utilization could be tagged as a CPU intensive task. This could be done empirically by calculating a threshold value from tasks to known with known bottlenecks on different resources.

Depending on the nature of how tasks are profiled and the information required by the scheduler to effectively designate tasks to nodes, it may be useful to propagate other information to the JobTracker such as network bandwidth, separate read and write statistics, etc. This would be rather trivial information to add provided that a means of obtaining this information from the operating system is available and can be added to the heartbeat without significantly increasing the heartbeats size.

B. Prevent Bandwidth Overutilization

Each additional byte added to the heartbeat is more traffic on the network. As the number of nodes scales up, this could become a potential issue as by default is transmitting 1 heartbeat/node/3 seconds. Currently, the resource utilization statistics consist of nine longs, 2 floats, and one int. This corresponds to 84 bytes of information. However, not all of this information may be necessary for the scheduler and it may be possible to trim this already small amount of data down even more. If any more statistics are added to heartbeat, care must be taken to ensure that the heartbeat does not become so large that it impedes other network traffic.

C. Advanced Sampling

The currently utilized sampling techniques, particularly for disk IO and CPU usage are fairly limited and may not be accurate enough to correctly profile the resource utilization of a node. There are a number of potential ways to improve this statistics. For instance, different Linux applications such as the top command could potentially be used to get a better idea of CPU utilization. It may also be worthwhile to have resource sampling run as a separate thread from the TaskTracker in order to poll the system more often for data.

D. Hadoop headed in this direction

When doing our first designs of Hadoop EKG we worked with Hadoop 0.20.2, which had no support for sending resource data over the heartbeat. However, when first attempted our implementation we decided to go with the Bleeding Edge version on Apache's GitHub account. Here, we found a number of additions to the TaskTracker to allow resource usage transmission over the heartbeat and a means of obtaining this information on a Linux operating system. While the JobTracker in the latest version of Hadoop does not actually utilize this information, the Hadoop developers have apparently seen value in obtaining and sending it.

We believe that this is an indicator that Hadoop is already moving in the direction of utilizing node resource information to more effectively schedule tasks. Also, a discussion in this JIRA Ticket brings up scheduling–At some point, a scheduler should also consider the resources available on the TT (mem, CPU) and use that to decide what combination of Map and Reduce slots should run on that node." [1]

VI. Conclusion

After evaluating the different methods to propagate real-time data from JobTrackers to the TaskTracker, we successfully implemented Hadoop EKG–a system that piggybacks heartbeats to propagate real-time resource utilization data. We propagate resource utilization data such as CPU usage, disk I/O and memory usage, and we find that this data can be very useful for the scheduler to make smart decisions scheduling tasks based on the real-time resource utilization information. Through the course of our work, we learned that Hadoop 0.22-alpha is already moving in this direction of propagating more resource data in heartbeats, and managed to build our system with this design choice in mind. Lastly, we believe the ideal next step is to use the resource data from the TaskTrackers to implement smart scheduling based on task resource utilization.

Acknowledgment

The authors would like to thank Nedyalko Borisov and Professor Shivnath Babu.

Appendix

Our source code is available at https://github.com/ganjianwei/hadoop_ekg.

TABLE I
Format of */proc/diskstats* File

Field	Value
1	Reads Issued
2	Reads Merged
3	Sectors Read
4	ms Reading
5	Writes Completed
6	Writes Merged
7	Sectors Written
8	ms Writing
9	I/Os in Progress
10	ms on I/Os
11	Weighted ms on I/Os

TABLE II
Format of */proc/stats* File (units in Jiffies)

Field	Value
1	Normal Processes
2	Niced Processes
3	System Processes
4	Idle
5	Waiting for I/O
6	Service Interrupts
7	Servicing Softirqs

References

[1] Hadoop jira ticket. https://issues.apache.org/jira/browse/HADOOP-3136.
[2] Philip Zeyliger. Cloudera blog: Hadoop metrics. http://www.cloudera.com/blog/2009/03/hadoop-metrics/, March 2009.