

# Tuning Hadoop map slot value using CPU metric

Kamal Kc Vincent W. Freeh

North Carolina State University

{kkc,vwfreeh}@ncsu.edu

## Abstract

Hadoop is a widely used open source mapreduce framework. Its performance is critical because it increases the usefulness of products and services for a large number of companies who have adopted Hadoop for their business purposes. One of the configuration parameters that influences the resource allocation and thus the performance of a Hadoop application is map slot value (MSV). MSV determines the number of map tasks that run concurrently on a node. For a given architecture, a Hadoop application has an MSV for which its performance is best. Furthermore, there is not a single map slot value that is best for all applications. A Hadoop application's performance suffers when MSV is not the best. Therefore, knowing the best MSV is important for an application. In this work, we find a low-overhead method to predict the best MSV using a new Hadoop counter that measures per-map task CPU utilization. Our experiments on a wide variety of Hadoop applications show that using a single MSV for all applications results in performance degradation up to 132% when compared to using the best MSV for each application.

## 1. Introduction

Hadoop is an open source mapreduce framework used by hundreds of companies for a variety of applications, which include indexing products in ecommerce webservices, log analysis, reporting, analytics, and machine learning [2]. The performance of Hadoop is important in increasing the usefulness of these products.

Performance tuning in Hadoop is a complex task as it has more than 150 configuration parameters that directly or indirectly affect its resource utilization and performance. The most common method for selecting best configuration values is trying several possible values and manually tweaking them until a Hadoop application completes in the least amount of time [1]. This process quickly becomes cumbersome and inefficient when finding best values for more than one Hadoop application. Thus, it is desirable to have a mechanism to select a best set of configuration parameters. In this work, we find a mechanism to predict the best value of a Hadoop parameter called *map slot value*.

Map slot value (MSV) is the maximum number of map tasks that run concurrently on a tasktracker node. Its misconfiguration can create significant performance degradation. We find that a Hadoop application has a unique MSV for which its performance is the best. Furthermore, there is not a single MSV that has the best performance for all Hadoop applications. For example, in one of our experimental clusters there are four MSVs that have the best performance for at least one application, but these MSVs have maximum performance degradation as high as 132%. Thus, it is important to know the best MSV for an application in order to avoid its performance degradation.

In this paper, we present a method to reliably predict the best MSV with low-overhead. Our method uses two new Hadoop counters that measure per-map task CPU utilization and IO throughput.

In the following sections of the paper, we present the related work, describe Hadoop map phases, describe our modifications, present the map phase completion time results, and describe the prediction of MSV.

## 2. Related work

Four areas of prior research are related to our work. The first area is optimizing configuration parameters of Hadoop. Research in this area explores different methods to obtain the optimal Hadoop configuration values. The methods include deriving the values from the optimal values of other jobs [9] or using metrics obtained from extensive instrumentation to extrapolate the optimal values [5, 7, 8]. Our work does not require knowing optimal values of other jobs or performing extensive instrumentation.

The second area of prior research is the optimization of resource utilization by selecting the best predefined technique. These approaches offer an important alternative method for optimizing performance behavior. They select the best technique by using rules [3], using program analysis [11] or measuring the completion time of several alternative implementations [13].

The third area of prior research is workload analysis of Hadoop jobs. These studies focus on creating standardized benchmarks [6] or creating qualitative job classes such as small, medium, and large duration jobs [10]. Our work can be extended to correlate the previously studied job classes and their performance behavior.

The fourth area that is closely related to our work is the research on Hadoop schedulers. This work does not focus on optimizing the resource utilization of the entire cluster, but rather focuses on optimizing resource utilization within the resource bounds imposed by the fixed preconfigured MSV [12, 14]. Our approach finds the best MSV, which ensures the efficient utilization of the cluster resources.

In addition to prior research, PUMA is also used in our work [4]. PUMA is a Hadoop benchmarking suite developed by Purdue University. PUMA includes three mapreduce programs from the official Hadoop distribution and ten other mapreduce programs. The collection of diverse applications makes it a useful benchmarking suite. In our experiments, we use six PUMA Hadoop applications. The combination of the six with our custom applications include the entire range of the measured metrics. In our future work, we plan to evaluate additional benchmark programs, including all those in PUMA.

## 3. Hadoop and modifications

A Hadoop application consists of *map* and *reduce* tasks. A map task processes a block of data and produces key-value output pairs. The map output is partitioned according to the range of the key. A reduce task aggregates and operates on the map output key-value pairs that fall under the assigned key range partition.

The map phase is usually the most time consuming operation of a Hadoop application. The applications used in our work, which

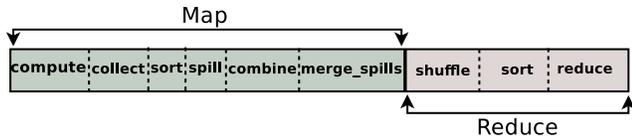


Figure 1. Map task phases.

represent common Hadoop applications, have an average map time of 67% of the total job runtime. The lowest map time for an application is 42% of the total job runtime whereas the highest map time is 85% of the total job runtime. Due to this reason, in this work we focus on optimizing the map runtime.

### 3.1 Map phases

A map task consists of 6 phases, which are compute, collect, sort, spill, combine, and *merge\_spills*. These phases are shown in Figure 1. In the compute phase, a map task applies the map function to each input key-value pair. In the collect phase, the map task stores the processed key-value pairs in a map output buffer. The sort phase occurs between collect and spill operations. In this phase, the output key-value pairs are sorted before the spilling occurs. When the map output buffer is full, the map task empties the buffer by spilling its content to a spill file in the local disk. This is the spill phase. The combine phase is optional and when present, the map task performs a local reduce operation on the map output key-value pairs. In the *merge\_spill* phase, the spill files are merged together to produce a single map output file.

Each phase in the map task is either CPU or IO intensive. The CPU intensive phases are compute, collect, sort, and combine. The IO intensive phases are spill and *merge\_spills*.

### 3.2 Modifications

Our approach predicts the best MSV using the CPU utilization and IO throughput metrics of a map task. The metrics are derived from the durations of the map phases. The best MSV is the MSV setting for which an application has the shortest completion time. An application has the best performance when all CPUs are fully utilized. Additionally, IO bound applications suffer performance degradation when the IO bandwidth is fully consumed. Thus, the best MSV setting either utilizes all CPUs efficiently or in case of an IO bound application fully utilizes the IO bandwidth. For MSVs greater than the best, the parallelism is too great, resulting in either CPU or IO performance degradation. The CPU performance degradation occurs due to additional system overhead for the larger number of processes. The IO performance degradation occurs due to higher contention for the IO bandwidth which lowers the overall IO throughput of the system. On the other hand, for MSVs lower than the best, the parallelism is low, resulting in either CPU or IO underutilization. When the CPU is underutilized, the CPU user and system time is low. When the IO resource is underutilized, there is leftover IO bandwidth available for use. In both cases, additional tasks can be run to use the leftover resources and improve the application completion time.

We use Hadoop counters to measure per-map task CPU utilization and IO throughput. Counters are built-in low-overhead metrics in Hadoop. They report important task and job related statistics. By default, a Hadoop task collects at least 16 statistical values using the counters. An example of a Hadoop counter is HDFS.BYTES.WRITTEN, which records the total amount of output data written to HDFS by reduce tasks.

We introduce two new counters CPU\_UTIL and IO\_THRPUT to measure per-map task CPU utilization and IO throughput. CPU\_UTIL is the sum of the time taken by the CPU intensive

Applications	CPU_UTIL(%)	IO_THRPUT(MB/s)
terasort	36	7.31
rankedinvertedindex	40	4.42
terasort(L10,D100)	46	4.79
wordcount	58	2.84
terasort(L30,D100)	58	3.87
invertedindex	65	1.99
terasort(L60,D100)	69	3.02
termvectorperhost	75	3.77
terasort(L100,D100)	77	2.29
terasort(L200,D100)	87	1.41
terasort(L500,D100)	94	0.65
grep	97	0.01
terasort(L10,D1)	97	0.11
terasort(L10,D0.1)	98	0.01
terasort(L10,D0.01)	99	0.01

Table 1. Utilization and throughput in the ascending order of CPU\_UTIL for PowerPC cluster.

phases of a map task, which are compute, collect, sort, and combine. IO\_THRPUT is the quotient of total map output bytes divided by the map task duration. The overhead of these counters is insignificant and is same as maintaining other existing Hadoop counters. Additionally, as map tasks have homogeneous CPU and IO behavior, we only need to measure the counter values of a map task instance to estimate the best MSV for an application.

## 4. Evaluation

In this section, we describe the experimental setup, analyze the performance of the Hadoop applications for different MSVs, and describe the prediction of MSV using the metric values.

### 4.1 Experimental setup

Experiments are performed on two clusters. The first cluster consists of six IBM PowerPC machines. Each node contains two POWER7 processors with 24 cores and 48 total CPU threads, 90GB RAM, and a 10 Gbps Ethernet network link. In the PowerPC cluster, Hadoop is configured with one jobtracker and five tasktrackers. HDFS is configured with one namenode and five datanodes. The second cluster consists of six x86 machines. Each node contains two Intel Xeon x86 processors with 8 cores and 16 total CPU threads, 24GB RAM, and a 10 Gbps Ethernet link. As in the PowerPC, in this cluster, Hadoop is configured with one jobtracker and five tasktrackers. HDFS is configured with one namenode and five datanodes.

Our experiments use fifteen Hadoop applications, among which six applications are from PUMA benchmark suite [4] and the remaining nine applications are customized versions of *terasort*. The PUMA applications are *grep*, *wordcount*, *invertedindex*, *rankedinvertedindex*, *terasort*, and *termvectorperhost*. The PUMA applications use wikipedia dataset. Terasort and its variants use the data generated by teragen.

We use the variants of *terasort* in order to explore the entire spectrum of CPU utilization and IO throughput values. Among the PUMA applications, the lowest CPU utilization of a map task is 36% for *terasort* and the highest is 97% for *grep*. Similarly, the lowest IO throughput is 0.01MB/s for *grep* and the highest is 7.31MB/s for *terasort*. However, the PUMA applications do not include all CPU utilization values between 36% and 97% or all IO throughput values between 0.01MB/s and 7.31MB/s. In order to include the entire utilization spectrum, in the *terasort* application we add variable number of extra busy loops and to include the entire IO

Data size=150GB, CPU cores per node=24, CPU threads per node=48							
Job	MSV(normalized)						
	16 (0.33)	24 (0.50)	32 (0.67)	40 (0.83)	48 (1)	56 (1.17)	64 (1.33)
terasort	1.17	<b>1(258s)</b>	1.02	1.18	1.89	2.32	2.51
rankedinvertedindex	1.30	1.09	1.01	<b>1(453s)</b>	1.11	1.09	1.23
terasort(L10,D100)	1.28	1.07	1.01	<b>1(350s)</b>	1.16	1.03	1.19
word count	1.57	1.26	1.26	1.06	1.08	<b>1(564s)</b>	1.04
terasort(L30,D100)	1.34	1.20	1.14	1.08	1.19	<b>1(470s)</b>	1.17
invertedindex	1.49	1.21	1.21	1.06	1.09	<b>1(620s)</b>	1.02
terasort(L60,D100)	1.19	1.13	1.15	1.11	1.14	<b>1(689s)</b>	1.13
termvectorperhost	1.38	1.16	1.16	1.02	1.08	<b>1(694s)</b>	1.02
terasort(L100,D100)	1.15	1.08	1.10	1.07	1.01	<b>1(948s)</b>	1.09
terasort(L200,D100)	1.16	1.10	1.10	1.10	1.13	<b>1(1539s)</b>	1.12
terasort(L500,D100)	1.15	1.08	1.11	1.09	1.06	<b>1(3459s)</b>	1.05
grep	1.18	1.05	1.05	<b>1(245s)</b>	1.11	1.39	1.40
terasort(L10,D1)	1.13	<b>1(173s)</b>	1.01	1.20	1.59	1.90	1.98
terasort(L10,D0.1)	1.22	<b>1(175s)</b>	1.05	1.11	1.47	1.62	1.84
terasort(L10,D0.01)	1.08	1.02	<b>1(190s)</b>	1.02	1.54	1.68	2.06
<b>Aggregate</b>	<b>18.79</b>	<b>16.45</b>	<b>16.38</b>	<b>16.10</b>	<b>18.65</b>	<b>19.03</b>	<b>20.85</b>
<b># of best values</b>	<b>0</b>	<b>3</b>	<b>1</b>	<b>3</b>	<b>0</b>	<b>8</b>	<b>0</b>

**Table 2.** Normalized performance and the best completion time (in parentheses) for different MSVs on the PowerPC cluster with 150GB dataset.

throughput spectrum we vary the amount of map output data. Table 1 shows that after adding the terasort variants, the applications include the entire spectrum of CPU utilization from 36% to 99% and IO throughput from 0.01 MB/s to 7.31MB/s. The terasort variants are listed by showing the number of busy loops and the amount of output data. For the terasort variant terasort(L10, D100), L represents the number of busy loops and D represents the percentage of input data that is converted to output. Thus, terasort(L10, D100) executes 10 extra busy loops for each key-value pair and outputs 100% of the input data. The highest number of busy loops is 500 and the lowest amount of output data is 0.01%. The variables L and D are used to control the CPU utilization and IO throughput of a map task. Increasing the value of L increases the amount of CPU computation performed by a map task. Additionally, decreasing the value of D decreases the amount of output data produced by a map task.

## 4.2 Performance analysis

Table 2 shows the normalized performance behavior of the fifteen applications for different MSVs and normalized MSVs for PowerPC cluster with 150GB data. The normalized MSV is the MSV relative to the number of CPU threads in a node.<sup>1</sup> The normalized MSVs are shown in parentheses alongside the MSVs in the header of the table. For the PowerPC machines, a normalized MSV of 1 means an actual MSV of 48, which is equal to 1 map task per CPU thread (or 2 map tasks per core). The best performance value is 1 and it denotes the shortest completion time of an application for the set of MSVs used in the experiments. The best MSV is the one for which an application has the shortest completion time. The shortest completion time is shown in parentheses for each application. In the PowerPC cluster, MSV is set to values from 16 to 64 with increments of 8. Below 16 and beyond 64, the applications suffer slowdown and those results are omitted.

<sup>1</sup>The machines used in our experiments have hyperthreading enabled due to which the CPU schedulable contexts as seen by operating system is greater than the number of cores. In a hyperthreaded system, the metric CPU\_UTIL measures the utilization of the threads, and a 100% utilization occurs when all threads are busy rather than when all cores are busy.

Table 2 shows that there is a best value for each application and there is not a single MSV that is best for all applications. Every application in the table has a unique best MSV. For example, terasort and wordcount have best MSVs of 24 and 56. Additionally, the table does not have a single MSV that is best for all applications. The last row shows the number of best values for different MSVs. MSVs 24, 32, 40, and 56 are best for 3, 1, 3, and 8 applications. One significant MSV is 40, which has lowest aggregate performance value of 16.10. The aggregate value is 7.3% higher than the theoretical best aggregate performance value of 15. But, it has a maximum slowdown of 20% for terasort(L10,D10). Thus, picking MSV 40 for all applications is not a best solution. Additionally, while MSV of 56 is best for the most applications, it has a slowdown of 132% for terasort. This further reinforces that a single MSV is not a best choice for all applications.

In order to test if these results are generally applicable, we also run these applications on both a larger dataset size and a different architecture (x86). Tables 3 and 4 show the normalized performance behavior of the PowerPC cluster for 300GB dataset and x86 cluster for 150GB dataset. The results show performance behavior similar to the PowerPC cluster with 150GB data size. In Table 3, MSVs 24, 32, 40, and 56 are best for at least one application. MSV 32 has the lowest aggregate performance value of 16.03, which is 6.9% higher than the theoretical best aggregate performance value of 15. It has a maximum slowdown of 13%. Similarly, in Table 4, MSVs 8, 12, and 16 are best for at least one application. As the number of cores and threads are different in the x86 cluster, for the experiment, MSV is set to values from 4 to 24 with increments of 4. Beyond 24, the applications suffer slowdown. MSV 12 has the lowest aggregate performance value of 16.09 which is 7.3% higher than the best aggregate performance value. It has a maximum slowdown of 16%. Thus, in the three cases tested, there is not a single MSV that is best for all applications.

## 4.3 Prediction

Table 1 shows an inverse linear relationship between CPU\_UTIL and IO\_THRPUT metrics. When CPU\_UTIL increases, IO\_THRPUT decreases and *vice versa*. Using linear regression of Table 1 data, we can approximate IO\_THRPUT for the PowerPC cluster. There-

Data size=300GB, CPU cores per node=24, CPU threads per node=48							
Job	MSV(normalized)						
	16 (0.33)	24 (0.50)	32 (0.67)	40 (0.83)	48 (1)	56 (1.17)	64 (1.33)
terasort	1.13	<b>1(519s)</b>	1.06	1.57	2.42	2.49	2.92
rankedinvertedindex	1.32	1.09	1.02	<b>1(771s)</b>	1.22	1.47	2.09
terasort(L10,D100)	1.27	1.09	<b>1(663s)</b>	1.03	1.42	1.14	1.73
word count	1.52	1.22	1.09	1.03	1.03	<b>1(1110s)</b>	1.02
terasort(L30,D100)	1.32	1.12	1.10	1.04	1.02	<b>1(939s)</b>	1.11
invertedindex	1.49	1.20	1.09	1.03	1.02	1.01	<b>1(1307s)</b>
terasort(L60,D100)	1.19	1.14	1.09	1.05	1.03	<b>1(1328s)</b>	1.09
termvectorperhost	1.35	1.18	1.12	1.07	1.02	<b>1(1368s)</b>	1.03
terasort(L100,D100)	1.17	1.15	1.11	1.04	1.03	<b>1(1800s)</b>	1.09
terasort(L200,D100)	1.16	1.14	1.13	1.07	1.05	<b>1(3030s)</b>	1.06
terasort(L500,D100)	1.14	1.13	1.11	1.09	1.05	<b>1(6814s)me</b>	1.07
grep	1.33	1.17	1.05	<b>1(513s)</b>	1.01	1.06	1.03
terasort(L10,D1)	1.08	<b>1(248s)</b>	1.01	1.02	1.20	1.17	1.23
terasort(L10,D0.1)	1.11	<b>1(238s)</b>	1.02	1.08	1.18	1.21	1.26
terasort(L10,D0.01)	1.11	<b>1(232s)</b>	1.04	1.09	1.24	1.23	1.28
<b>Aggregate</b>	<b>18.69</b>	<b>16.63</b>	<b>16.03</b>	<b>16.24</b>	<b>17.94</b>	<b>17.78</b>	<b>20.01</b>
<b># of best values</b>	<b>0</b>	<b>4</b>	<b>1</b>	<b>2</b>	<b>0</b>	<b>7</b>	<b>1</b>

**Table 3.** Normalized performance the best completion time (in parentheses) for different MSVs on PowerPC cluster with 300GB datasize.

Data size=150GB, CPU cores per node=8, CPU threads per node=16						
Job	MSV(normalized)					
	4 (0.25)	8 (0.50)	12 (0.75)	16 (1)	20 (1.25)	24 (1.5)
terasort	1.14	<b>1(2688s)</b>	1.05	1.11	1.34	1.46
rankedinvertedindex	1.09	<b>1(1676s)</b>	1.11	1.24	1.35	1.44
terasort(L10,D100)	1.13	<b>1(2172s)</b>	1.09	1.21	1.31	1.44
word count	1.88	1.04	<b>1(1337s)</b>	1.17	1.12	1.21
terasort(L30,D100)	1.39	1.12	<b>1(2670s)</b>	1.05	1.11	1.42
invertedindex	2.36	<b>1(509s)</b>	1.09	1.41	1.95	2.13
terasort(L60,D100)	1.24	1.24	1.01	<b>1(2732s)</b>	1.17	1.41
termvectorperhost	1.56	1.10	1.09	<b>1(536s)</b>	1.13	1.20
terasort(L100,D100)	1.61	1.29	1.14	<b>1(2871s)</b>	1.23	1.40
terasort(L200,D100)	1.54	1.18	1.13	<b>1(3114s)</b>	1.11	1.21
terasort(L500,D100)	1.85	1.25	1.16	<b>1(3960s)</b>	1.09	1.27
grep	1.57	1.12	<b>1(352s)</b>	1.05	1.09	1.10
terasort(L10,D1)	1.04	<b>1(883s)</b>	1.05	1.08	1.06	1.07
terasort(L10,D0.1)	1.03	<b>1(884s)</b>	1.03	1.09	1.11	1.13
terasort(L10,D0.01)	1.07	<b>1(871s)</b>	1.08	1.10	1.12	1.13
<b>Aggregate</b>	<b>21.5</b>	<b>16.34</b>	<b>16.09</b>	<b>16.51</b>	<b>18.29</b>	<b>20.02</b>
<b># of best values</b>	<b>0</b>	<b>7</b>	<b>3</b>	<b>5</b>	<b>0</b>	<b>0</b>

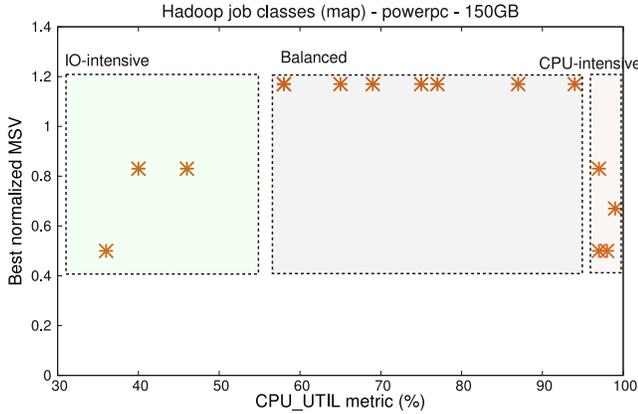
**Table 4.** Normalized performance the best completion time (in parentheses) for different MSVs on x86 cluster with 150GB datasize.

fore, while searching for a metric to predict best MSV for a Hadoop application, we only use the values of CPU\_UTIL. Figure 2 shows the normalized best MSVs for the CPU utilization combinations of all 15 applications.

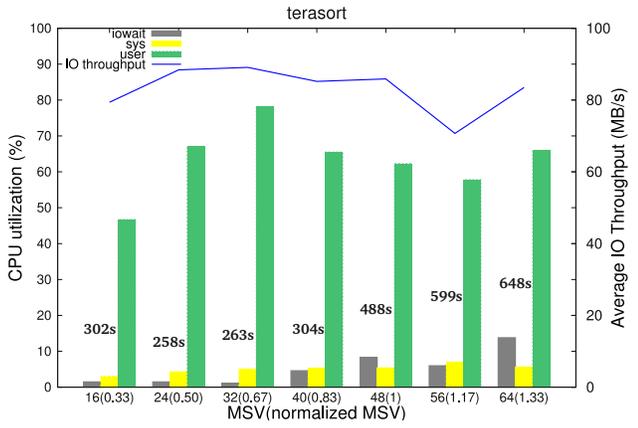
In Figure 2, the applications are divided into three general regions: *IO-intensive*, *Balanced*, and *CPU-intensive*. Each region is based on a different range of CPU\_UTIL and a predictable range of best MSV. *IO-intensive* region has applications with low CPU\_UTIL (36% to 60%). This region is called *IO-intensive* because applications in this region have high IO\_UTIL. The normalized best MSVs of applications in this region are less than 1.0. *Balanced* region has applications with medium CPU\_UTIL (60% to 90%). The normalized best MSVs of applications in this region are greater than 1.0. *CPU-intensive* region has applications with

high CPU\_UTIL (90% to 100%). Applications in this region have low IO\_UTIL. The best normalized MSVs of applications in this region are less than 1.0. The normalized best MSVs greater than 1.0 means that the number of map tasks exceeds the number of CPU threads in the system. This indicates that the applications are scalable. On the other hand, the best normalized MSVs less than 1.0 indicate that the applications are not scalable and face resource bottlenecks. The scalable nature of *Balanced* region and the bottlenecks of *IO-intensive* and *CPU-intensive* regions are described with examples in the following paragraphs.

Figures 3, 4, and 5 show the CPU and IO behavior of a task-tracker node for an application of each region. The figures show the node behavior including the completion time for all MSVs and help to explain the performance of an application when MSV is the



**Figure 2.** Classification for applications running on PowerPC cluster.



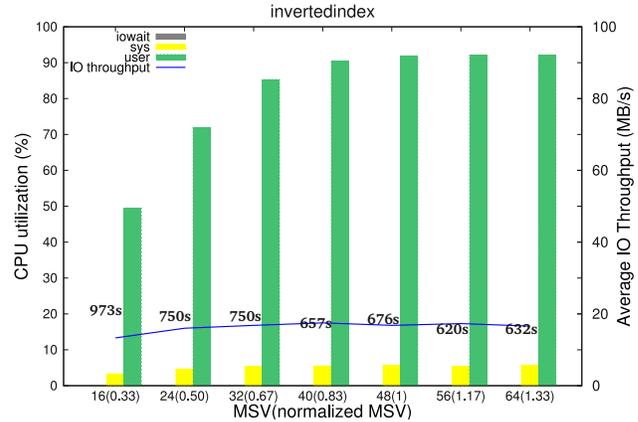
**Figure 3.** CPU utilization, IO throughput, and completion time (shown in relative vertical heights) of an *IO-intensive* application (terasort).

best. In the figures, the CPU utilization is divided into *user*, *system*, and *iowait* states. The *user* is the time spent by the map tasks, the *system* is the time spent by the kernel, and the *iowait* is the time spent waiting for the IO operation to complete. Each figure is described as follows.

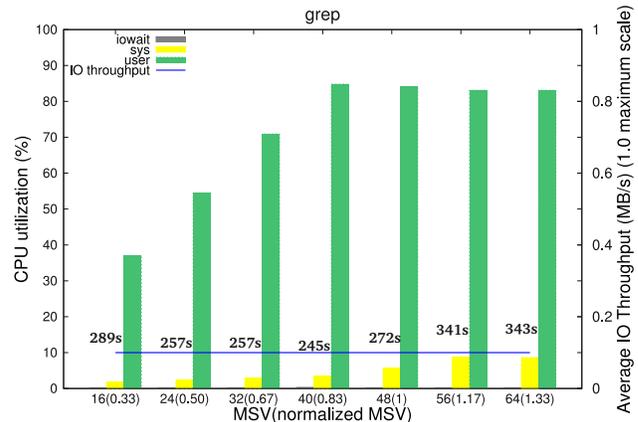
Figure 3 shows the node behavior of *terasort*, which falls in *IO-intensive* region. The figure shows 24 as the best MSV. In Figure 3, the IO throughput increases until MSV is 24. For MSV greater than 24, the IO throughput decreases and there is an increase in *iowait*. This suggests that beyond 24 MSV, increasing parallelism merely increases IO pressure and overhead due to the IO pressure. This explains the lower relative best MSV for applications in *IO-intensive* region.

Figure 4 shows the node behavior of *invertedindex*, which falls in *Balanced* region. The figure shows 56 as the best MSV. In Figure 4, the *user* CPU increases until MSV is 56. After 56, the *user* CPU levels off without showing performance improvement. IO throughput on the other hand is almost constant and does not peak, which suggests a lack of IO bottleneck. Due to this reason, there is not any noticeable *iowait*. This behavior results in the applications having best MSV greater than the number of CPU threads.

Figure 5 shows the node behavior of *grep*, which falls in *CPU-intensive* region. The figure shows 40 as the best MSV, which is



**Figure 4.** CPU utilization, IO throughput, and completion time (shown in relative vertical heights) of a *Balanced* application (invertedindex).



**Figure 5.** CPU utilization, IO throughput, and completion time (shown in relative vertical heights) of a *CPU-intensive* application (grep).

83% of the total number of virtual CPU threads. In Figure 5, the *user* CPU steadily increases until MSV is 40. After 40, the *sys* CPU increases and the *user* CPU levels off and decreases in small amount. This suggests that due to the high CPU utilization of *grep*, the system overhead increases. The IO throughput on the other hand is 0.1MB/s for all MSVs.

**Prediction for x86 cluster.** The prediction for x86 cluster is similar to the PowerPC, except the difference in region boundaries. The IO throughput of a x86 node and a PowerPC node is 40MB/s and 100MB/s respectively.<sup>2</sup> Due to this reason, applications with medium CPU\_UTIL, which in turn have medium IO\_THRPUT, suffer from IO bottleneck in the x86 cluster whereas they do not suffer from IO bottleneck in the PowerPC cluster. As a result, the applications with medium CPU\_UTIL belong to *IO-intensive* region of x86 cluster instead of *Balanced* region. Additionally, as these applications with medium IO\_THRPUT have relatively higher CPU\_UTIL values, the *Balanced* region starts at a higher CPU\_UTIL value in the x86 cluster. This is observed in Figure 6, which shows the best normalized MSV for all CPU\_UTIL values.

<sup>2</sup> A node in the x86 cluster has a single SAS hard disk, whereas a PowerPC node has 5 SAS hard disks in RAID-5 configuration.

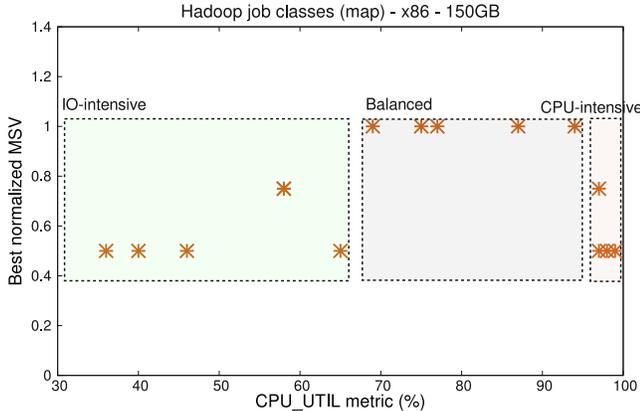


Figure 6. Classification for applications running on x86 cluster.

MSV selection	PowerPC		x86
	150GB	300GB	150GB
Best single MSV	16.10(20%)	16.03(13%)	16.09(16%)
Predicted MSV using the regions	15.19(6%)	15.21(6%)	15.50(11%)

Table 5. Aggregate normalized performance values and maximum slowdown percentages (inside parentheses) for the three tested cases.

From the figure, we observe that the *Balanced* region starts at 69% which is relatively higher than 58% for PowerPC.

**Performance characteristics of using predicted MSV.** To find the effectiveness of the regions, we compare the performance of applications when using the region specific MSVs and a best single MSV for all applications. For the fifteen applications, the best aggregate normalized performance value is 15. For the *IO-intensive*, *Balanced*, and *CPU-intensive* regions, we select the normalized MSVs 0.67, 1.17, and 0.67 for the PowerPC cluster and 0.75, 1, and 0.75 for the x86 cluster. The best single MSV for PowerPC is 0.67 and for x86 it is 0.75. Table 5 shows the performance values for these two schemes. Using a single predicted MSV for applications in each region has a better aggregate performance value. For the three cases, the aggregate slowdown when using region based predicted MSV compared to when using a single fixed MSV is 1.2% and 7.3%, 1.4% and 6.8%, and 3.3% and 7.3% respectively. In the parentheses alongside the performance values, the table shows in percentage the maximum slowdown of applications when using the predicted MSVs. It gives an upper bound of slowdown for an application, which occurs during the worst case scenario. By using the region based predicted MSVs, the maximum slowdown decreases to as little as 6% from 20% when compared to using a single best MSV.

In this section, we showed the performance results of Hadoop applications for two clusters: PowerPC and x86, and two data sizes for PowerPC cluster: 150GB and 300GB. Our findings show that we can predict the performance behavior based on the CPU\_UTIL metric. The performance characteristics fall into three general regions: *IO-intensive*, *Balanced*, and *CPU-intensive*. The *IO-intensive* region contains applications with high IO throughput and low CPU utilization and the normalized best MSV is below 1. The *Balanced* region contains applications with medium CPU utilization and medium IO throughput and the normalized best MSV is 1 or above. The *CPU-intensive* region contains applications with high CPU utilization and low IO throughput and the best MSV is

below 1. The CPU utilization or IO throughput value at which the regions separate differs depending upon a node’s hardware characteristics. For a new application, CPU\_UTIL measurement indicates the region the application belongs to and based on that region the MSV can be set to the predicted value.

## 5. Conclusion

Optimizing resource allocation to improve performance in Hadoop is an important area of research. Improved Hadoop performance adds value to hundreds of Hadoop deployments in commercial as well as research organizations. In this work, we explored the performance behavior of fifteen Hadoop applications that included wide range of CPU and IO characteristics. We observed that each Hadoop application has a unique MSV for which it has the best performance. MSV is a Hadoop configuration parameter for the maximum number of map tasks that run concurrently on a task-tracker node. Additionally, there is not a single MSV that is best for all applications. Based on these findings, we developed a method to predict the best MSV. Our method uses a new Hadoop counter that measures per-map task CPU utilization. The results showed that based on the counter value the applications form three distinct regions. Each region’s application has a specific range of MSV that results in its best performance. When using the region based predicted MSVs, the aggregate performance degradation is only 1%, which is comparatively less than 7% when using a single MSV for all applications. Furthermore, the slowdown for any application is as low as 6% when using region based prediction compared to 20% when using a single MSV. Thus, the low-overhead method of using metric values to predict MSV is an efficient approach for estimating the best configuration parameter value and achieving the best performance for Hadoop applications.

## References

- [1] Avoiding common hadoop administration issues. <http://blog.cloudera.com/blog/2010/08/avoiding-common-hadoop-administration-issues>.
- [2] Hadoop poweredby. <http://wiki.apache.org/hadoop/PoweredBy>.
- [3] Hadoop vaidya. <http://hadoop.apache.org/docs/stable/vaidya.html>.
- [4] Faraz Ahmad, Seyong Lee, Mithuna Thottethodi, and T. N. Vijaykumar. Puma: Purdue mapreduce benchmarks suite. *Technical Report, Purdue University*, 2012.
- [5] Shivnath Babu. Towards automatic optimization of mapreduce programs. In *Proceedings of the ACM symposium on Cloud computing (SOCC)*, 2010.
- [6] Yanpei Chen, Sara Alspaugh, and Randy Katz. Interactive analytical processing in big data systems: a cross-industry study of mapreduce workloads. In *Proc. of VLDB*, 2012.
- [7] H. Herodotou and S. Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. In *Proc. of the 37th International Conference on Very Large Data Bases (VLDB)*, 2011.
- [8] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *Proc. of Conference on Innovative Data Systems Research (CIDR)*, 2011.
- [9] Karthik Kambatla, Abhinav Pathak, and Himabindu Pucha. Towards optimizing hadoop provisioning in the cloud. In *Proceedings of the 2009 conference on Hot topics in cloud computing*. USENIX Association, 2009.
- [10] Asit K. Mishra, Joseph L. Hellerstein, Walfredo Cirne, and Chita R. Das. Towards characterizing cloud backend workloads: insights from google compute clusters. In *Proc. of SIGMETRICS*, 2010.
- [11] Christopher Olston, Benjamin Reed, Adam Silberstein, and Utkarsh Srivastava. Automatic optimization of parallel dataflow programs.

In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, 2008.

- [12] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguade, M. Steinder, and I. Whalley. Performance-driven task co-scheduling for mapreduce environments. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, 2010.
- [13] Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the atlas project. *Parallel Computing*, 2000.
- [14] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proc. of EuroSys*, 2010.