Optimal Tradeoff Between Energy Consumption and Response Time in Large-scale MapReduce Clusters

Pavlos Paraskevopoulos
Dept. of Informatics,
Aristotle University of Thessaloniki, Greece
Email: paparask@csd.auth.gr

Anastasios Gounaris
Dept. of Informatics,
Aristotle University of Thessaloniki, Greece
Email: gounaria@csd.auth.gr

***Abstract*—The increasing growth of the size of the digital databases has given rise to the need for the development of infrastructures, such as large scale data centers and computational clusters, which are capable of storing and processing very large volumes of data. To date, most clusters have been designed for performance. Due to non-linear speed-ups that are common to typical applications, performance maximization involves the decision of the number of the nodes to process a specific (intensive) task, as opposed to the utilization of the full cluster. In addition, energy consumption has recently attracted significant attention, given that the cost to operate a cluster may well exceed its acquisition cost. This issue calls for judicious use of resources as well. The aim of this study is to present a method that achieves the optimal tradeoff between energy consumption and response time in distributed clusters, such as MapReduce clusters. To this end, we propose an algorithm that derives the fraction of the nodes that minimizes the energy consumption without sacrificing performance (in terms of response time) more than a user-defined threshold. Moreover, we present a generic and configurable framework to describe performance and energy consumption as a function of the nodes used; our framework can accommodate the widely spread MapReduce-like parallel executions in a straightforward manner. The evaluation results show that our methodology can lead to significant energy savings with acceptable performance penalty in many realistic situations.**

***Keywords*-energy consumption, response time, map-reduce**

I. Introduction

Large-scale clusters typically consist of a high number of nodes that can run in parallel with a view to speeding-up data- or computation-intensive processes. However, there are several barriers to linear speed-up, such as process initialization and interference overheads [1], which may even lead to performance degradation if the number of the cluster nodes used exceeds an application-dependent value [2].

Another challenge in achieving efficient parallelism, apart from the potential performance degradation, stems from the energy consumption. According to [3], direct monthly energy cost makes up 23% of the total monthly costs that a data center needs for the servers, the power and the infrastructure cost. The cost for the cooling and the power distribution make up 82% of the infrastructure cost. Consequently, the total energy cost is a significant portion of the cluster's operating cost. Furthermore, considering that server costs are falling, it is estimated that, by 2012, the three year cost of electricity per server will exceed the initial cost of the server purchase by 3 to 22 times [4]. In addition, recent studies show that energy inefficiency problems are aggravated when the nodes are under-utilized [5], i.e., they are not used at their full potential. According to the studies in [5] and taking into account the under-utilization of the machines when they are in cluster [3], a large portion of the total monthly operating cost due to power consumption can be reduced if we increase energy efficiency of the nodes during low utilization periods. This can be achieved by keeping some nodes inactive (i.e., powered down) rather than idle during process execution.

In this paper, we present a method that achieves the optimal tradeoff between energy consumption and response time in distributed clusters, such as MapReduce clusters. To this end, we propose an algorithm that derives the fraction of the nodes that minimizes the energy consumption without sacrificing performance (in terms of response time) more than a user-defined threshold. More specifically, the contributions of our paper are as follows:

- We present a generic and configurable framework to describe response time and energy consumption as a function of the nodes used; our framework can accommodate the widely spread MapReduce-like parallel executions in a straightforward manner emphasizing on the way speed-up is achieved and the initialization overheads.
- We present an algorithm that achieves the optimal tradeoff between the response time and the energy consumption, in the sense that energy consumption is minimized while the maximum performance degradation does not exceed a user-specified threshold.
- We evaluate our methodology and the results show that our approach can lead to significant energy savings with acceptable performance penalty in many realistic situations, whereas there are cases in which both response time and energy consumption are minimized.

The structure of this paper is as follows. In Sec. 2, we briefly discuss the studies on which our method is based and other related work. In Sec. 3, we present our algorithm and the accompanying framework. The evaluation is in Sec. 4, and Sec. 5 concludes the paper.

II. Related Work

Many techniques and frameworks to handle and process data in a distributed system have been developed. In this work, we concentrate on distributed systems that parallelize tasks according to the MapReduce framework (with some extensions). MapReduce is basically a programming model, the implementation of which (e.g., Hadoop) renders parallelization and fault-tolerance issues transparent to the user [6]. In MapReduce applications, a specific node plays the role of the coordinator, which assigns workload to the working nodes and supervises the execution. An interesting application of MapReduce to generic data management tasks, namely multi-way joins that appear in several applications, is described in [7]. Two features of the work in [7] include i) the fact that the optimal parallelization of some tasks may lead to sublinear speed-ups even if there are no barriers due to task initialization overheads, and ii) some applications may include parts that cannot be parallelized. In our work, we consider these interesting cases explicitly.

Nowadays, most of the MapReduce algorithms that have been proposed focus on performance only thus ignoring energy consumption issues. However, there exist some studies that take into account both the energy consumption and the response time (e.g., [8], [9], [10]). In [8], two methods that target at the minimization of the energy consumption by activating or deactivating the nodes are presented. The main idea behind these methods is that, by deactivating some nodes, utilization and energy efficiency is improved thus leading to lower energy consumption. The first method uses the minimum set of the nodes, which may lead to serious performance degradation. The second method utilizes all the nodes in the cluster, and, upon the completion of task execution, powers down the nodes to avoid the high energy consumption at idle state. Several powered down states are examined in [8] from which hibernation seems the most efficient in terms of the trade-off between energy consumption and transition time (from online to offline state and vice versa). Our approach can be regarded as a hybrid of these two methods with the addition that we consider the case where high degrees of parallelism may not be beneficial. Also, the energy consumption functions in our work are based on those in [8].

The work in [10] performs energy aware-resource allocation by matching hardware resource usage characteristics to process requirements. Energy consumption is considered indirectly by maximizing resource utilization, which is known to be more energy efficient. Finally, a query optimizer that takes into consideration both the execution time and the energy consumption is presented in [9].

Table I
Notation

N	Number of available nodes in the cluster
k	Number of online (non powered-down) nodes
$T(k)$	Response time
$E(N,k)$	Energy consumed
P	Power cost of an active node (per time unit)
T_{tr}	Time to transition a node from offline to online state (or vice versa)
P_{tr}	Power cost to transition a node from offline to online state (or vice versa)
P_h	Power cost during hibernate (offline) state (per time unit)
a	Parallelizable cost (in time units)
b	Initialization cost per node (in time units)
c	Non-parallelizable cost (in time units)
s	Skew parameter
δ	Performance degradation parameter

III. Our Approach

As explained above, the main rationale of our algorithm is to power down some nodes with a view to minimizing energy consumption while the increase of response time is below a threshold. In other words, we trade performance for higher energy efficiency, when these goals are contradictory. Table I summarizes the notation used.

A. Algorithm Description

Let $T(k)$ be the response time of a task that employs k nodes in a cluster with N nodes ($1 \leq k \leq N$). The algorithm to compute the tradeoff between response time and energy consumption is shown in Fig. 1. In the first step of algorithm, the number of nodes that minimizes the response time is computed. If $T(k)$ is (strictly) monotonical, then utilizing all the nodes in the cluster leads to minimal response time. However, in practice $T(k)$ is unimodal, and approaches such as golden section search can be employed to detect the extremum value of k and the value range. The average-case complexity of such approaches is logarithmic, i.e., the approaches are extremely fast. The second step of the algorithm computes a range of k values, for which the performance degradation does not exceed a parameter δ. Finally, the value of k within this range that minimizes the energy consumption is computed and the algorithm finishes. The output of the algorithm defines the exact number of the machines to be used, whereas all the other nodes are powered down, e.g., to the hibernate state, which is characterized by low transition time and low energy consumption as explained in [8]. Because of the unimodality of the function, the tradeoff between energy and response time given a specific degradation threshold is optimal; any solution outside the range of values examined is inferior.

B. Framework for Estimating Response Time and Energy Consumption

1) Response Time: We assume that the nodes in the cluster are homogeneous with regards to their technical characteristics. We divide each task's cost into two parts, namely the parallelizable cost a and the non-parallelizable

Algorithm: int FindOptimalTradeoff (N, δ)

```
1. find $k'$ s.t. $T(k)$ is minimized;
2. find range $[k'_{min}, k'_{max}]$, $k'_{min} \leq k' \leq k'_{max}$
       s.t. $T(k) \leq (1+\delta)T(k')$
3. find $k \in [k'_{min}, k'_{max}]$ s.t. $E(N,k)$ is minimized
4. return $k$
```

Figure 1. Outline of the algorithm.

cost c. Furthermore, each node selected to participate in the task execution is initialized in b time units. According to these, the response time is given by the following equation:

$$T(k) = \frac{a}{f(k)} + b \cdot g(k) + c \tag{1}$$

$f(k)$ defines the speed-up of the parallelizable part of the task (without considering initialization and other types of overheads). Simple tasks achieve linear speed-up (LS), which corresponds to the case where $f(k) = k$. However, there are several cases, in which the optimal speed-up is sublinear. In [7], a case is described, for which $f(k) = \sqrt[3]{k^2}$. We will use this case as a representative of sublinear speed-up (SLS) cases.

$g(k)$ defines the aggregate impact of node initialization on the task response time. We consider two common cases. First, when nodes are initialized by a coordinator in a sequential manner before the beginning of execution. In this case, which will be referred to as serial initialization (SI), $g(k) = k$. The response time can be decreased if the initialization phases of all nodes overlap, e.g., when the nodes form a (potentially temporary) hierarchy during initialization phase according to which each node is responsible for initializing its children at the next hierarchy level. Such hierarchical initialization (HI) is described by a logarithmic function $g(k) = ln(k)$.

Eq. (1) can be easily extended to capture the cases in which the system is not fully balanced, e.g., because of skew in workload distribution. In the following equation, we have inserted parameter $s,\ 1/k \leq s \leq 1$. Smaller values correspond to totally unbalanced scenarios (i.e., all the work is allocated to a single node), whereas $s = 1$ denotes that workload is equally distributed.

$$T(k) = \frac{a}{s \cdot f(k)} + b \cdot g(k) + c \tag{2}$$

Overall, there are four cases:

LS-HI: $T(k) = \frac{a}{s \cdot k} + b \cdot ln(k) + c$
SLS-HI: $T(k) = \frac{a}{s\sqrt[3]{k^2}} + b \cdot ln(k) + c$
LS-SI: $T(k) = \frac{a}{s \cdot k} + b \cdot k + c$
SLS-SI: $T(k) = \frac{a}{s\sqrt[3]{k^2}} + b \cdot k + c$

2) Energy Consumption: We assume that, initially, all nodes in the cluster are online. As such, $N - k$ nodes are powered down, since non-fully utilized machines result in both energy and performance inefficiency if they are kept online. For each machine that is powered down, we spend $P_{tr} \cdot T_{tr}$ power to change their state; after this transition, we spend P_h for the whole duration of the task execution. The total energy that the system consumes during execution is:

$$E(N,k) = k \cdot P \cdot T(k) + (N-k) \cdot (P_{tr} \cdot T_{tr} + P_h \cdot T(k)) \tag{3}$$

The first term describes the energy consumption attributed to the online nodes, while the second term gives the energy consumption of the offline nodes. Note that more sophisticated equations can be derived that take into account several offline states and the overlap of transition time of offline nodes and the execution time; such extensions are not examined here and are left for future work.

¿From Eq. (3), it can be observed that the ratio between the power costs at different states is a key factor. According to [8], we can make the assumption that P_{tr} is similar to the power P when the cluster is fully on and running a workload. However, T_{tr} is defined by the capabilities of the hardware and the operating system; in general, we expect this to be orders of magnitude smaller that the total response time of intensive tasks. If T_{tr} is relatively high, the delays to change the state of a node may become unacceptable. Finally, our approach makes more sense when P_h is significantly smaller than P, in order to yield energy benefits. In practice, there is a trade-off between P_h and T_{tr}, and modern systems support several powered down states with different levels of trade-offs.

IV. Evaluation

In this section, we evaluate our approach by showing how our algorithm can yield significant energy savings with low performance penalty. Essentially, we compare our solution against approaches that target optimal performance regardless of the energy consumption.

The default characteristics of the cluster are as follows: P = P_{tr} = 10, P_h = 1, T_{tr} = 0.1. According to these parameters, the power cost per time unit for online nodes is the same as the cost to power down a node and an order of magnitude higher than the offline cost; this is a realistic assumption as explained earlier. Initially, we assume that the task is parallelized in a balanced manner (s=1) and the maximum performance degradation allowed is 10% ($\delta = 0.1$). We experiment with all four types of parallelization (namely LS-HI, LS-SI, SLS-HI, SLS-SI). The parameters $a,\ b$ and c are chosen in such a way that the parallelizable cost may differ from 0 (for non intensive tasks) to 2 orders of magnitude (for intensive tasks). More specifically, $a,\ b$ and c can take two values each: $a = 10, 100,\ b, c = 1, 10$. We also assume that the cluster consists of either 100 or 10000 nodes. The combination of these parameters correspond to cases where the cluster is both under-utilized and fully utilized.

We first examine the LS-HI case. For each case, we investigate all $2^3 = 8$ combinations of $a,\ b$ and c for

Table II
LS-HI (FOR N=10000)

a	b	c	k_t	k_e	$[k'_{min}, k'_{max}]$	k
10	1	1	10	10	[5,29]	10
10	1	10	10	9	[3,92]	9
10	10	1	2	2	[2]	2
10	10	10	2	2	[2]	2
100	1	1	100	72	[39,413]	72
100	1	10	100	57	[26,1189]	57
100	10	1	10	10	[5,26]	10
100	10	10	10	10	[5,29]	10

Table III
LS-HI (FOR N=100)

a	b	c	k_t	k_e	$[k'_{min}, k'_{max}]$	k
10	1	1	10	4	[5,29]	5
10	1	10	10	3	[3,92]	3
10	10	1	2	2	[2]	2
10	10	10	2	2	[2]	2
100	1	1	100	14	[39,100]	39
100	1	10	100	9	[26,100]	26
100	10	1	10	5	[5,26]	5
100	10	10	10	4	[5,29]	5

$N = 10000$ (Table II) and $N = 100$ (Table III). In each table, we show i) the value of k that minimizes $T(k)$, which is computed at the first step of the algorithm and is denoted by k_t, ii) the value of k that minimizes $E(N, k)$ regardless of any performance degradation, which is denoted by k_e, iii) the range $[k'_{min}, k'_{max}]$ which is computed at the second step of the algorithm, and iv) the final value of k found by the algorithm.

If our system has 10000 nodes, the cluster is basically under-utilized and the values of k_t, k_e and k are either identical or very close to each other, i.e., energy and performance are both (near-)optimal, apart from the cases where a and b differ by two orders of magnitude. In the latter cases, which is also the most commonly encountered in real-world intensive tasks, the output of our algorithm k equals k_e, which is significantly lower than k_t. For $c = 1$, our algorithm employs 72 nodes instead of 100 (benefitting from 28 machines being powered down). The energy savings are greater when $c = 10$, where 57 nodes instead of 100 are employed. When $N = 100$, the cluster may become fully utilized. In these cases (where again a and b differ by two orders of magnitude), k is higher than k_e, but still significantly lower than k_t, which means that the energy cost can be reduced by several factors (e.g., from 100 machines down to only 26) at the expense of tolerable time cost.

Interestingly, for LS-SI, the energy benefits are less significant even for intensive jobs. This is due to the fact that, for the parameters chosen, the optimal level of parallelism according to both criteria is particularly low, whereas our algorithm is more suitable for large scale parallelism. The results are shown in Tables IV (for $N = 10000$) and V (for $N = 100$).

In SLS-HI (see Tables VI and VII for $N = 10000$ and $N = 100$, respectively), we encounter cases where the

Table IV
LS-SI (FOR N=10000)

a	b	c	k_t	k_e	$[k'_{min}, k'_{max}]$	k
10	1	1	3	3	[2,4]	3
10	1	10	3	3	[2,6]	3
10	10	1	2	2	[2]	2
10	10	10	2	2	[2]	2
100	1	1	10	10	[7,15]	10
100	1	10	10	10	[6,17]	10
100	10	1	3	3	[3,4]	3
100	10	10	3	3	[2,5]	3

Table V
LS-SI (FOR N=100)

a	b	c	k_t	k_e	$[k'_{min}, k'_{max}]$	k
10	1	1	3	3	[2,5]	3
10	1	10	3	2	[2,6]	2
10	10	1	2	2	[2]	2
10	10	10	2	2	[2]	2
100	1	1	10	7	[7,15]	7
100	1	10	10	6	[6,17]	6
100	10	1	3	3	[3,4]	3
100	10	10	3	3	[2,5]	3

Table VI
SLS-HI (FOR NODES=10000)

a	b	c	k_t	k_e	$[k'_{min}, k'_{max}]$	k
10	1	1	17	16	[6,75]	16
10	1	10	17	14	[4,252]	14
10	10	1	2	2	[2]	2
10	10	10	2	2	[2]	2
100	1	1	544	166	[138,3938]	166
100	1	10	544	117	[87,10000]	117
100	10	1	17	16	[7,64]	16
100	10	10	17	15	[6,75]	15

Table VII
SLS-HI (FOR NODES=100)

a	b	c	k_t	k_e	$[k'_{min}, k'_{max}]$	k
10	1	1	17	4	[6,75]	6
10	1	10	17	2	[4,100]	4
10	10	1	2	2	[2]	2
10	10	10	2	2	[2]	2
100	1	1	100	12	[67,100]	67
100	1	10	100	8	[52,100]	52
100	10	1	17	4	[7,64]	7
100	10	10	17	4	[6,75]	6

Table VIII
SLS-SI (FOR N=10000)

a	b	c	k_t	k_e	$[k'_{min}, k'_{max}]$	k
10	1	1	3	3	[2,5]	3
10	1	10	3	3	[2,6]	3
10	10	1	2	2	[2]	2
10	10	10	2	2	[2]	2
100	1	1	12	12	[8,21]	12
100	1	10	12	12	[7,22]	12
100	10	1	3	3	[2,5]	3
100	10	10	3	3	[2,5]	3

minimum response time is achieved if several hundreds of nodes are used. The behavior is similar to the case of LS-HI with the difference that the energy savings are more significant. For example, our algorithm may employ up to nearly five times less nodes (117 instead of 544) for intensive

Table IX
SLS-SI (FOR N=100)

a	b	c	k_t	k_e	$[k'_{min}, k'_{max}]$	k
10	1	1	3	2	[2,5]	2
10	1	10	3	2	[2,6]	2
10	10	1	2	2	[2]	2
10	10	10	2	2	[2]	2
100	1	1	12	6	[8,21]	8
100	1	10	12	6	[7,22]	7
100	10	1	3	2	[2,5]	2
100	10	10	3	2	[2,5]	2

Table X
SLS-HI (FOR N=10000, T_{tr}=1)

a	b	c	k_t	k_e	$[k'_{min}, k'_{max}]$	k
10	1	1	17	16	[6,75]	16
10	1	10	17	14	[4,252]	14
10	10	1	2	2	[2]	2
10	10	10	2	2	[2]	2
100	1	1	544	176	[138,3938]	176
100	1	10	544	120	[87,10000]	120
100	10	1	17	16	[7,64]	16
100	10	10	17	16	[6,75]	16

Table XI
SLS-HI (FOR N=10000, S=0.8)

a	b	c	k_t	k_e	$[k'_{min}, k'_{max}]$	k
10	1	1	24	21	[8,110]	21
10	1	10	24	18	[5,366]	18
10	10	1	2	2	[2,3]	2
10	10	10	2	2	[2,3]	2
100	1	1	761	195	[188,5759]	195
100	1	10	761	137	[119,1000]	137
100	10	1	24	21	[8,95]	21
100	10	10	24	21	[8,110]	21

Table XII
SLS-HI (FOR N=10000, δ=0.05)

a	b	c	k_t	k_e	$[k'_{min}, k'_{max}]$	k
10	1	1	17	16	[8,46]	16
10	1	10	17	14	[5,99]	14
10	10	1	2	2	[2,3]	2
10	10	10	2	2	[2,4]	2
100	1	1	544	166	[197,2030]	197
100	1	10	544	117	[137,3992]	137
100	10	1	17	16	[9,42]	16
100	10	10	17	15	[8,46]	15

parallelizable tasks. Again, if the full cluster should be utilized to achieve minimum response time, k_e is lower than k, but still, the energy consumption is decreased. For SLS-SI, the same observations as in the LS-SI apply (Tables VIII and IX).

In the last part of the evaluation, we investigate the effects of T_{Tr}, s and δ. Due to lack of space, we present results only for the SLS-HI case. The key findings are summarized as follows. Even if there is a ten-fold increase in T_{tr}, the impact on the algorithm's efficiency is very small (see Table X). The energy savings may increase even further in unbalanced intensive executions (see Table XI), whereas a decrease in δ to the half has small impact on energy consumption (see Table XII).

V. Conclusion

In this paper, we presented an algorithm that achieves an optimal tradeoff between the response time and the energy consumption in large-scale clusters. Our algorithm computes the portion of the cluster that should be powered down in order to maximize energy savings without increasing the response time above a threshold. Our algorithm is particularly effective in settings, where the following conditions apply: (i) the parallelizable cost is at least an order of magnitude higher than the non-parallelizable one (note that, if this is not the case, running the task in a large-scale cluster is questionable); and (ii) the parallelizable cost is at least two orders of magnitude higher than the node initialization cost. Both these conditions are commonly encountered in real-world massively parallel (e.g., MapReduce-based) applications. Our work can be extended in several ways, the most important of which include the direct comparison against [8] and the consideration of multiple offline states.

References

[1] D. J. DeWitt and J. Gray, "Parallel database systems: The future of high performance database systems." *Commun. ACM*, vol. 35, no. 6, pp. 85–98, 1992.

[2] A. N. Wilschut, J. Flokstra, and P. M. G. Apers, "Parallelism in a main-memory DBMS: The performance of PRISMA/DB." in *VLDB*, 1992, pp. 521–532.

[3] J. Hamilton, "Cooperative expendable micro-slice servers (cems): Low cost, low power servers for internet-scale services," in *CIDR*, 2009.

[4] K. G. Brill, "Data center energy efficiency and productivity," in *The Uptime Institute - White Paper available at http://www.uptimeinstitute.org*, 2007.

[5] L. A. Barroso and U. Hölzle, "The Case for Energy-Proportional Computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI '04*, 2004, pp. 137–150.

[7] F. N. Afrati and J. D. Ullman, "Optimizing joins in a map-reduce environment," in *Proceedings of the 13th International Conference on Extending Database Technology*, 2010, pp. 99–110.

[8] W. Lang and J. M. Patel, "Energy management for mapreduce clusters," *Proc. VLDB Endow.*, pp. 129–139, 2010.

[9] W. Lang, R. Kandhan, and J. M. Patel, "Rethinking query processing for energy efficiency: Slowing down to win the race," in *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2011.

[10] W. Xiong and A. Kansal, "Energy efficient data intensive distributed computing," in *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2011.