Energy-efficient scheduling on milliclusters with performance constraints

Frédéric Pinel, Johnatan E. Pecero and Pascal Bouvry
Computer Science and Communications Research Unit
University of Luxembourg
firstname.lastname@uni.lu

Samee U. Khan
Dept. of Electrical and Computer Engineering
North Dakota State University, Fargo, ND, USA
samee.khan@ndsu.edu

***Abstract*—Today's datacenters and large scale enterprise computing are power hungry. A lot of research effort is devoted in industry and academy to address this challenging issue. In this context, a new type of enterprise computing platform is being investigated. This computing platform is composed of hundred of millicomputers, each requiring orders of magnitude less power. The millicomputing aims to significantly increase the energy efficiency of data centers by delivering high performance computing. However, this approach brings challenges that must be met in order to compete with the current practice. This paper addresses two such critical challenges. First, it suggests how to decompose large applications into smaller tasks, better suited to millicomputers. Then, it casts the performance oriented and energy efficient problem into a soft real-time scheduling problem, for which several algorithms are then proposed and evaluated. Sensitivity analysis is used to provide insights into the model, and plan the evaluation of the scheduling algorithms. The contention found in multi-core millicomputing processors is also accounted for.**

I. Introduction

Energy efficiency is justified by the rising energy costs in the data center [1]. The components responsible for the majority of the electrical costs are the server machines used in data centers. These costs come from the direct power consumption of the servers, but also indirectly from the cooling equipment required to keep the data center operational [1]. The combined costs for the direct and indirect power consumption can amount to 75% of the total energy costs in a data center.

The current server machines are based on high performance multi-core processors, where the number of cores is steadily increasing. Despite the recent improvements in energy-efficiency of these processors, they remain power hungry and dissipate significant heat. This is aggravated by the recent trend to package these machines in so-called blades, themselves grouped in racks. This aggravation is due to cooling costs not growing linearly with the temperature [2].

Millicomputing [3] is an alternative computing platform for data centers. It suggests to use computing elements, such as processors, that require much less power than those found in conventional servers. Because the individual performance of these low power machines is much less than the current data center servers, they need to be grouped into clusters in order to collectively provide enough performance. Clusters of millicomputers are called milliclusters [3].

The millicomputing initiative raises several issues, addressed in this paper. How to design applications for millicluster machines, which also scale to the many -distributed- cores available? How can a millicluster keep the low power property of its millicomputers, when attempting to meet performance objectives comparable to those found in data centers? This paper is a preliminary investigation of the millicluster alternative.

The contributions of this paper are:

- proposes an application architecture that allows large applications to run on a millicluster,
- proposes and evaluate several heuristics for the energy efficient use of resources in a millicluster, while meeting application performance objectives,
- includes the effect of contention in multi-core processors, such as those found in millicomputers,
- analyzes the models introduced with sensitivity analysis.

This paper is organized as follow. Section II presents milliclusters. Section III details how to design applications for this distributed platform. Section IV provides the energy model considered in this paper. Scheduling algorithms, sensitivity analysis of the models and experimental results are described in Section V. Section VI concludes the paper.

II. Millicomputing

This section describes millicomputing and its challenges.

A. Milliclusters

The computing platform considered in this paper is based on millicomputing [3]. It investigates the issue of rising energy costs by replacing the components responsible for the majority of the cost of energy with, existing, more energy-efficient and less heat producing equivalents.

The idea is to turn to the market of mobile phones, smartphones and other mobile computing devices, for solutions to the increasing cost of energy in data centers. Mobile computing devices have, by specification, to successfully address this issue, and it is fair to say they have mostly succeeded. The mobile device industry has designed components, including processors, that can consume milliwatts (hence the name millicomputing), as opposed to the hundreds of watts of traditional servers. Furthermore, these devices do not require cooling.

The processors suitable for a millicomputer cannot deliver a performance comparable to that of a typical server processor. Therefore, it is suggested to group several of these energy efficient devices into a small distributed system, which is called a *millicluster*.

B. Multi-core processors

It is important to note that the millicomputer processors can be multi-core, but with a smaller number of cores than typically found in data center servers.

Multi-core processors are not true parallel machines, in the sense that significant resources are shared between cores. The resulting contention considerably impacts the performance of a task [4], [5], [6], [7], [8], [9], [10], and indirectly its energy consumption.

For example, current many-core processors propose the view of an unified and shared memory. This leads to complicated memory path design which still requires great care from the programmer. Yet, this increased programming effort provides diminishing returns in terms of performance [11], [12].

Although milliprocessor processors provide less cores than their data center server counterparts, it is necessary to account for the contention, because of its impact on performance and energy. This aspect is included in this paper.

III. Achieving performance objectives in a millicomputing system

The use of millicomputers raises a challenging question: how to deliver equivalent performance to the current data center servers, when relying or considerably slower processors?

Milliclusters (employing many slower processors) also introduce the reputed difficulties associated with distributed systems.

A. Pipelining

Large applications must be split in smaller parts, which are then distributed across the milliprocessor machines in hope of matching performance of current multi-core processors.

The chosen message-based decomposition is software pipelining (also called stream processing), introduced by D. Mc Ilroy in Unix shells. A software pipeline is a set of tasks, connected via uni-directional communication links. The advantage of a pipeline is to increase throughput, at the cost of increased delay for completing all the steps in the pipeline. Therefore pipelines can provide better overall application performance, if this performance is based on throughput. Applications which rely on throughput versus latency may be servers handling many concurrent requests, where the latency for each request is less critical than the number of requests served. This includes many Internet services, which can face considerable load. High performance computing often produces highly parallel applications, where the latency of the processing of each task contributes less to the overall completion time than the number of concurrent processes.

Decomposing existing applications into software pipelines is not straigthforward. The application becomes distributed across the millicluster, which also introduces some complexity. This paper does not aim to indicate how to perform such decomposition. However, the widespread use of piped shell commands on the Unix operating systems, and the various distributed queue-based applications (such as workflows) provide examples of successful pipelined software constructions.

B. Application model

An application is decomposed into a set of *tasks*, connected by message queues. Tasks are arranged in a pipeline, a sequence of tasks, where each output is the input of the next task in line. Several copies of the same tasks may be run concurrently, all serving the same queue. However, this possible set up is not investigated further here.

The objective is to match the performance of the application on data center class processors. However, the application overall performance is now distributed across the tasks. Each task must respect some service level, in order for the application to do as well.

Soft real-time deadlines provide a natural way to express this constraint. Each task is therefore given a deadline. The overall pipeline performance is equal to the slowest task's performance. The performance of a task is modeled as a time needed to process an unit of input, called the estimated time to complete (ETC) [16]. Therefore, the deadline for each task is the largest ETC. All tasks share the same deadline.

In practice, that deadline is extended by a factor, $delta$, to account for variability in the ETC of the slowest task.

IV. Energy model

The motivation behind millicomputing is energy saving. Therefore, operating a millicluster should also aim to minimize energy costs. Although milliclusters do not require cooling, they should still minimize the energy used by their components.

In fact, the motivation for energy saving is cost saving. Reducing the computing infrastructure reduces costs. Total energy consumption for the system is related to the overall size of a computing infrastructure. Therefore total energy consumption is a good indicator of the overall operating costs for a computing infrastructure.

This section presents the model for energy consumption in a millicluster.

A. Machine energy estimation

The cores are packaged in processors, which are themselves grouped in a computing machine. This paper supposes that each core is capable of dynamic voltage frequency scaling (DVFS); that is, it can be operated on a set of supply voltages and different speed performance (associated to different clock frequencies) [17], [18]. DVFS seeks to exploit the convex relationship between the core supply voltage (that impacts the speed of execution) and the energy consumption. Moreover, different cores of a same processor are assumed able to independently operate at different voltage/frequency points.

Energy-efficiency using DVFS often considers that application could specify the voltage/frequency point of operation of

each core. This is not the assumption in this paper. Indeed, inspection of the kernel power management tools of the GNU/Linux kernel version `2.6.35-24`, reveals that DVFS is very dynamic and self-regulated. Default values for the On-demand governor show a sampling rate of 10 ms (time period when a DVFS change is considered). Fundamentally, the complexity of a cluster is such that whenever possible, local decision making should be preferred over a global one. In this case, the regulation is based on CPU utilization, which is also under the control of the kernel.

Here, the operating system (OS) manages power using the `cpu-freq` tools under the on-demand governor. The on-demand governor implements the *race-to-idle* policy. Whenever there is a need for CPU, then the voltage/frequency is set to its maximum value. Later, when the utilization decreases, the voltage/frequency point of operation is chosen so as to match the needed load. It should be mentioned that the automatic adjustment of the CPU frequency results in jitter, delayed comunications, in a large-scale system, which is the case of the millicluster.

Processor is not the only component of the millicluster to consume energy. Because the intention is to capture the total energy consumption of the millicluster, other components need to be included. The power model adopted is summarized by the relation:

$$P_m = P_{constant} + P_{high}, \tag{1}$$

where P_m is the total power of a machine (millicomputer), $P_{constant}$ represents the constant power term, for components which do not scale according to voltage or frequency, this also include the network, and P_{high} represents the power increase, compared to the idle state, when components subject to DVFS are in high performance mode (in an active state, the machine dissipates $P_{constant} + P_{high}$). This is automatically adjusted by the OS and the hardware, and does not only include the processor.

This model is preferable because it lends itself to experimental validation through power measurements.

Energy is the product of power and time. The energy should also reflect the race-to-idle policy. Total energy is defined here by:

$$E_m = P_{constant} \times CT_{max} + P_{high} \times \sum_{c}^{cores} CT_c, \tag{2}$$

where CT_c is the sum of all ETC of the tasks assigned to a core of a machine (its finishing time), and CT_{max} is the maximum CT_c over all cores of all machines. If no tasks are run on a machine, then that machine is considered switched off. When running a task, the core is at maximum voltage/frequency consuming P_{high}, when idle, only $P_{constant}$. The idle time lasts until the last core finishes its tasks.

All machines in the milliclusters are considered identical in this study. Usually, clusters are assembled at a given point in time, and machines are bought together.

B. Impact of contention in multi-core machines

As discussed in Section II-B, the millicomputers part of the millicluster are multi-core, which suffer from contention.

The contention considered in this study is related to components shared across a machine, such main memory. The contention factor is proportional to the memory parts of the other tasks running on the other cores of the same machine. Tasks running on the same core are not subject to contention because they do not run concurrently, but are preempted by the OS.

Contention effects impact the ETC of a task (access of a shared resource are serialized). Each task is defined by an ETC, which is split into:

- ETC under possible contention,
- the rest of the ETC, which is independent of sources of contention.

This effect of contention on a task's ETC is approximated by a delay added to its ETC. This penalty delay is the sum over all the other cores of the machine, of the time period spent concurrently in contention prone activity.

This definition requires the schedule of execution for each task on every core, which is the topic of the next section.

V. SCHEDULING PIPELINES IN MILLICLUSTERS

As mentioned at the end of the previous section, efficiently running software pipelines on a milliclusters of multi-core processors requires the precise scheduling of the tasks onto the cores.

Before describing the algorithms proposed in this paper, which is the topic of the next section, the nature of the scheduling in the millicluster must be made precise. Indeed, various components in a computing system use schedulers, making the term scheduling ambiguous.

The scheduler investigated here is a non-privileged instance of a program, operating at the cluster level. The main activity of the millicluster scheduler is to periodically define the set of concurrent tasks in a processor, so as to minimize contention, meet task deadlines and save energy.

It is similar to an OS long term scheduler, described in [19] by: "The long-term scheduler deals with the high-level or "big picture" issues; it is invoked infrequently and tasked with deciding which processes should inhabit the ready queue. The idea is that the long-term scheduler takes on the role of load balancing: it might try to maintain a mix of I/O bound (i.e., those that perform mainly I/O) and computationally bound processes for example, in order to give the best overall system performance."

The millicluster scheduler defines the current list of processes, per machine, that a usual OS scheduler (or short term) schedules for execution at a much higher frequency. The core on which to run the process is not important, because the millicomputers are considered single processor (yet multi-core), and the exact core mapping is not important (as opposed to the concurrent set of processes).

A. Algorithms

In this section, three scheduling algorithms are presented to meet the stated objectives.

The particular context for the scheduling question lies at the intersection of two fields:

- distributed system, for the scheduling independent tasks (tasks in a pipeline have become independent tasks, which is another benefit of this program structure),
- soft real-time, because of the deadlines for each task.

The first algorithm evaluated is a variant of Min-Min [20], [21], a mapping algorithm for resource allocation. It originates from the field of distributed systems. It is a greedy algorithm, which considers all possible task-to-core mappings, and then performs the assignment of the best mapping, constructing the schedule incrementally. "Best" is defined here by the energy delay product (EDP) $D \cdot E$, where:

- D is the total time by which all tasks exceeded their deadlines, $D = \sum_t^{tasks} max(0, deadline - finishing\_time)$,
- E is the total energy spent.

To influence the decisions of the algorithm, a paramater α is introduced, such that a mapping is considered better than the current best Max, if $\alpha \cdot D \cdot E < Max$.

The other two novel algorithms we propose are inspired from real-time scheduling algorithms: rate monotonic and earliest deadline first [22]. These algorithms have been proven optimal in a specific context, which is not the one set here, but similar. These variants are called Shortest Slack First (SSF) and Longest Slack First (LSF).

SSF orders the tasks to schedule in increasing slack time, the difference between their deadline and ETC. It then assigns a core (which minimizes the EDP according the same rule as Min-Min) for each task in turn, incrementally constructing the schedule.

LSF works identically, but orders the tasks by decreasing slack.

Before these algorithms can be experimentally compared, all the parameters of the different models (task, machine, energy, contention, algorithm) need to be set. It is best done once their respective influence is established. This is the topic of the next section.

B. Sensitivity analysis

Sensitivity analysis (SA) [23] of a model provides many benefits. First it determines the influence of each of the factors of the model. This allows future users of the model to focus on the most important parameters of the model, while ignoring the least influential.

It also helps design models, because understanding the influence of each factor allows to verify of the model. For example, SA allowed the authors to uncover an error in the earlier version of a model. The SA reported that a presumed key factor had in fact almost no influence.

The objective for this SA is Factors Prioritization (FP), whose goal is [23] "to make a rational bet on what is the factor that one should fix to achieve the greatest reduction in the uncertainty of the output". A factor corresponds to our model parameters.

TABLE I
MODEL PARAMETER VARIATION

Parameter	Probability	Range of values
maximum ETC	uniform	15 – 30 [time unit]
Contention rate max	uniform	30 – 60%
Deadline $delta$	uniform	0 – 30%
Power (constant)	uniform	15 – 25 W
Power high	uniform	0.1 – 2.0 W
α	uniform	0.7 – 1.2

The different parameters for our model are presented in Table I, with their range of possible values. Parameter maximum ETC is the maximum value used for the random generation of ETCs of all tasks. The maximum contention rate is the maximum value for the proportion of the ETC the task spends in contention prone instructions. This value is used for the generation of ETCs. Deadline is the additional time added to the largest ETC to obtain the common deadline. It is an additional percentage to the largest ETC. Powers have been defined in Section IV-A. The values for P_{high} is taken from the specifications of the ARM 9 processor. The values for $P_{constant}$ are arbitrarily chosen, to reflect the non-processor related components. Parameter α is part of the objective function defined in Section V-A. A small value for α indicates that sub-optimal scheduling decisions are allowed, whereas greater than 1 values indicate that scheduling decisions improve the objective by a greater margin.

Two SA are performed: a quantitative and a qualitative method.

The quantitative method used is an extension to the Fourier Amplitude Sensitivity Test [24]. This method allows the computation of first order effects and interactions for each parameter. Parameters interaction occurs when the effect of the parameters on the output is not a sum of their single (first order) effects. This variance decomposition method has the following desirable properties [23]. It is model independent (it does not place requirements on the type of model to work). It evaluates the effect of a parameter while all others are also varying. Finally, it copes with the influence of scale and shape (the probability density function and its parameters).

The results for this method are shown in Figures 1, 2.

The output in the case of performance is the amount of time by which the deadlines were exceeded. Part of the hypothesis for the SA was a small number of machines, such that most tasks failed to meet their deadline. The amount of time by which they failed to meet their deadlines is the output.

Regarding the performance analysis, the factors have predominantly a linear impact on the output. The most influent factor is α. Maximum ETC, the deadline factor and maximum contention playing minor roles. Indeed, the definition of α makes it play an influential role. Maximum ETC sets the maximum value for the task generation. The ETC for each task is randomly chosen in a given range. A higher value for

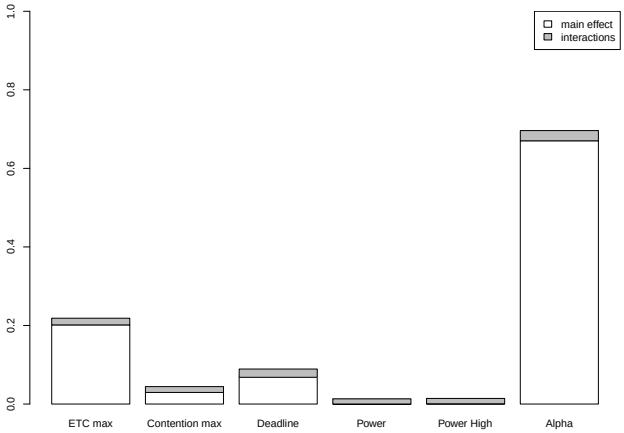

Fig. 1. Fast99 of Performance

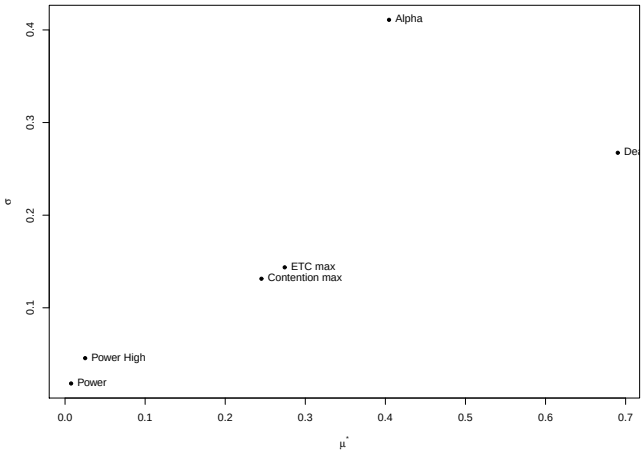

Fig. 3. Morris of Performance

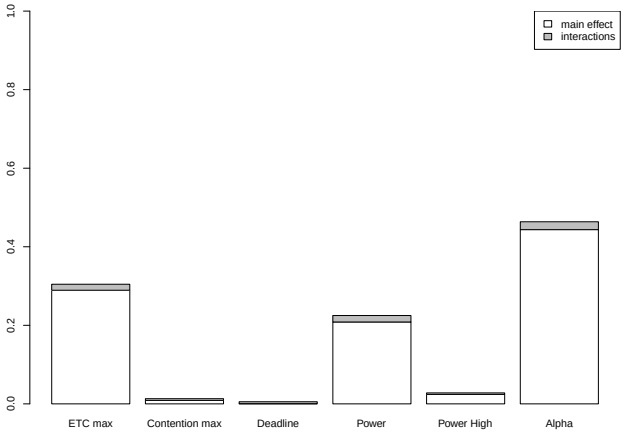

Fig. 2. Fast99 of Energy

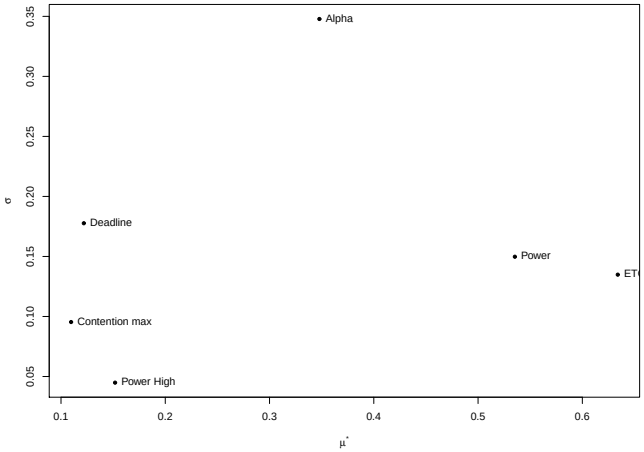

Fig. 4. Morris of Energy

maximum ETC leads to heterogeneous task ETC.

Regarding the energy analysis, the three important factors are α, maximum ETC and power. This is interesting because it exposes the tight relation between performance and energy.

The qualitative method used is a ”one factor at a time” (OAT). It is commonly used for screening the least important factors from the rest. The method used here is the method of Morris. It also captures the linear and non-linear interaction of factors. Qualitative methods require less computations than quantitative ones.

The results for the method of Morris are shown in Figures 3, 4. The x-axis indicates the linear impact of the factor, while the vertical axis indicates the non linear impact.

Regarding the performance analysis, deadline and α show the strongest impact. Maximum ETC and maximum contention play a minor role. Both SA methods find the same four important factors, but in different order of importance.

Regarding the energy analysis, maximum ETC, power and α have the most influence on the output. This is identical to the results of Fast 99.

C. Comparison of the scheduling heuristics

This section compares the three scheduling heuristics presented in Section V-A.

The SA results show that special care should be taken when setting the parameters α, maximum ETC and to a lesser extent deadline and maximum contention. Therefore, experiments ran the different heuristics on the same ETC instances (30 instances for each run). Table II lists the parameter settings.

The choice for the machines are based on ARM 9 processors. The power values come from the hardware specifications.

Figure 5 presents the performance results for the three heuristics. The x-axis lists different values for α, the heuristic score parameter. Performance is the amount of time by which the deadlines were exceeded. All heuristics reach their best score when $\alpha \simeq 1$. LSF is the best algorithm for performance, but only slightly better than Min-Min. However, it is faster than Min-Min.

TABLE II
MODEL PARAMETERS FOR HEURISTIC COMPARISON

Parameter	Value
# tasks	16
# machines	4
# processor/machine	1
# cores/processor	2
ETC range	5 – 25 [time unit]
Contention range	10 – 60%
Deadline $delta$	10%
Power (constant)	20 W
Power high	1.0 W

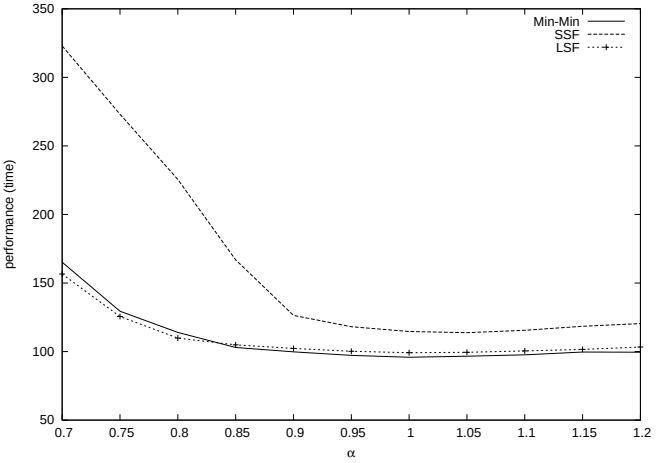

Fig. 5. Comparison of 3 heuristics on performance

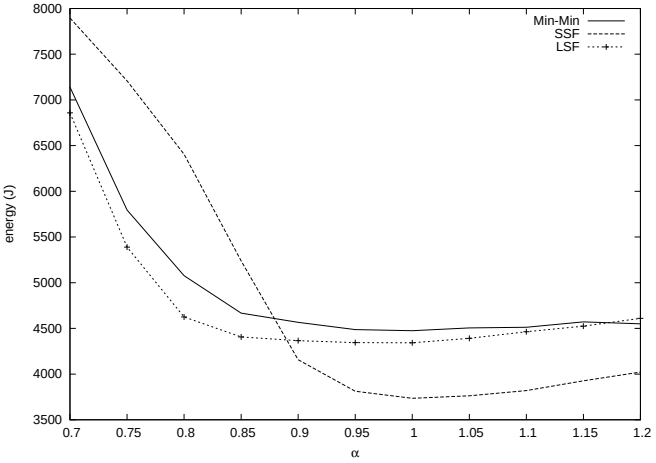

Fig. 6. Comparison of 3 heuristics on energy

Figure 6 presents the total energy results for the three heuristics. All heuristics reach their best score when $\alpha = 1$. SSF is the best algorithm for energy, and LSF is slightly better than Min-Min.

VI. CONCLUSION

In this paper, milliclusters along with their specific challenges were presented.

Software pipeline was introduced as a powerful application architecture to overcome the limited individual performance of each millicomputer, and allows to benefit from a low energy system.

Three scheduling algorithms were introduced and evaluated to further reduce the energy consumption while meeting performance objectives, by adding soft real-time considerations to the software pipeline. The contention in multi-core processors was also part of the model.

Finally, this was accomplished by applying two different sensitivity analysis methods.

Future work will focus on the validation of the results by experimenting on a real millicluster.

REFERENCES

[1] "Report to congress on server and data center energy efficiency public law 109-431," U.S. Environmental Protection Agency, ENERGY STAR Program, Tech. Rep., 2007, http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf.

[2] G. Varsamopoulos, A. Banerjee, and S. K. S. Gupta, "Energy efficiency of thermal-aware job scheduling algorithms under various cooling models," in *IC3*, ser. Communications in Computer and Information Science, S. Ranka, S. Aluru, R. Buyya, Y.-C. Chung, S. Dua, A. Grama, S. K. S. Gupta, R. Kumar, and V. V. Phoha, Eds., vol. 40. Springer, 2009, pp. 568–580.

[3] A. N. Cockcroft, "Millicomputing: The coolest computers and the flashiest storage," in *Int. CMG Conference*. Computer Measurement Group, 2007, pp. 407–414.

[4] D. Snowdon, S. Ruocco, and G. Heiser, "Power management and dynamic voltage scaling: Myths and facts," 2005.

[5] K. Klues, V. Handziski, C. Lu, A. Wolisz, D. Culler, D. Gay, and P. Levis, "Integrating concurrency control and energy management in device drivers," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, ser. SOSP '07. New York, NY, USA: ACM, 2007, pp. 251–264.

[6] F. . Pinel, J. . Pecero, P. . Bouvry, and S. . Khan, "Memory-aware green scheduling on multi-core processors," in *Proceedings of the Second International Workshop on Green Computing*, 2010, pp. 0–9.

[7] A. Mandal, R. Fowler, and A. Porterfield, "Modeling memory concurrency for multi-socket multi-core systems," in *ISPASS*. IEEE Computer Society, 2010, pp. 66–75.

[8] D. Pase, "The pchase benchmark page," http://pchase.org/.

[9] A. Fedorova, S. Blagodurov, and S. Zhuravlev, "Managing contention for shared resources on multicore processors," *Commun. ACM*, vol. 53, pp. 49–57, February 2010.

[10] R. West, P. Zaroo, C. A. Waldspurger, and X. Zhang, "Online cache modeling for commodity multicore processors," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, ser. PACT '10. New York, NY, USA: ACM, 2010, pp. 563–564.

[11] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006. [Online]. Available: \url{http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html}

[12] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, pp. 20–24, March 1995. [Online]. Available: http://doi.acm.org/10.1145/216585.216588

[13] A. S. William Gropp, Ewing Lusk, *Using MPI*. MIT Press, 1999.

[14] J. Armstrong, "The development of erlang," in *Proceedings of the second ACM SIGPLAN international conference on Functional programming*, ser. ICFP '97. New York, NY, USA: ACM, 1997, pp. 196–203. [Online]. Available: http://doi.acm.org/10.1145/258948.258967

[15] J.-R. Abrial, *Modeling in Event-B*. Cambridge University Press, 2010.

[16] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous," *Journal of Science and Engineering, Special 50 th Anniversary Issue*, vol. 3, pp. 195–207, 2000.

[17] J. R. Lorch and A. J. Smith, “Improving dynamic voltage scaling algorithms with pace,” *SIGMETRICS Perform. Eval. Rev.*, vol. 29, pp. 50–61, June 2001.

[18] V. Venkatachalam and M. Franz, “Power reduction techniques for microprocessor systems,” *ACM Computing Survey*, vol. 37, no. 3, pp. 195–237, 2005.

[19] D. Page, *A Practical Introduction to Computer Architecture*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[20] O. H. Ibarra and C. E. Kim, “Heuristic algorithms for scheduling independent tasks on nonidentical processors,” *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, 1977.

[21] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, “A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems,” *J. Parallel Distrib. Comput.*, vol. 61, pp. 810–837, June 2001. [Online]. Available: http://portal.acm.org/citation.cfm?id=511973.511979

[22] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, “Optimization and approximation in deterministic sequencing and scheduling: A survey.”

[23] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto, *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. Wiley, 2004.

[24] A. Saltelli, S. Tarantola, and K. Chan, “A quantitative, model independent method for global sensitivity analysis of model output,” *Technometrics*, vol. 41, pp. 39–56, 1999.