WORKLOAD CHARACTERIZATION OF MULTITHREADED APPLICATIONS ON MULTICORE ARCHITECTURES

Davide Cerotti
DEIB
Politecnico di Milano
via Ponzio 51
20133, Milano, Italy
davide.cerotti@polimi.it

Marco Gribaudo
DEIB
Politecnico di Milano
via Ponzio 51
20133, Milano, Italy
marco.gribaudo@polimi.it

Mauro Iacono
DSP
Seconda Università di Napoli
viale Ellittico 31
81100 Caserta, Italy
mauro.iacono@unina2.it

Pietro Piazzolla
DEIB
Politecnico di Milano
via Ponzio 51
20133, Milano, Italy
pietro.piazzolla@polimi.it

KEYWORDS

multicore; multithreading; performance modeling

ABSTRACT

Multicore architectures are now available for a wide range of high performance applications, ranging from embedded systems to large scale servers deployed in cloud environments. Multicore architectures are usually subject to two conflicting goals: obtaining a full utilization of the cores while achieving given performance objectives, such as throughput, response time or reduced energy consumption. Moreover, there is a strong interdependence between the software characteristics of the applications, and the underlying CPU architecture. In this scenario, simulation and analytical techniques can provide solid tools to properly design the considered class of systems: however, properly characterize the workload on multithreaded application in multicore environment is not an easy task, and thus is an hot research topic. In this paper we present several models, of increasing complexity, that can characterize multithreaded applications running on multicore architectures.

I. INTRODUCTION

Since the commercialization of the first microprocessor, the advancement of integration techniques allowed the implementation of a higher and higher transistor density on silicon dies. The increasing abundance of transistors on a single chip enabled the consumer market to adopt advanced architectural solutions, that progressively exploited more stages in pipelines, more functional units per stage, more control units per CPU and finally more cores per die. The availability of low cost multicore CPUs has been a key factor in shaping modern massively distributed computing systems, as the use of multicore CPUs allows: power consumption optimization, heating reduction, parallelism flexibility, process granularity exploitation, spatial packing enhancement. Virtualization technologies finally opened the way to the implementation of cheap, massively shared and virtualized computing facilities, obtained by means of (basically) off-the-shelf components that can easily and affordably be replaced in case of damages.

While multicore architectures and multithreading introduce significative advantages in terms of potential overall performances enhancements, they also introduce an additional complexity level that should be taken in account when designing performance models. The effects are specially relevant when dealing with critical systems or real time applications, but they also have to be considered whenever their impact is multiplied by the existence of a big number of instances that are active in a system at the same time, as in the case of massively distributed architectures or Big Data infrastructures.

In this paper the effects of multicore and multithreading are modeled to explore their relevance with respect to their impact as components of more complex architectures. The study is performed by means of analytical and simulative techniques.

The paper is structured as follows: the next section presents the general approach, then the subsequent section gives a glance of the state of the art; two more sections consider the modeling problem by the analytical and simulative point of view, by introducing new complexity elements in the model; conclusions end the paper.

II. MOTIVATION

Predictive performance modeling is a valuable support in the design and maintenance processes of computer based systems. Traditionally, many analytical and simulative techniques are applied, such as Petri nets or queuing networks, or event-based simulation.

An increase in complexity of the systems to be evaluated translates into an increase in complexity of the corresponding models. As far as the structure of a system keeps simple, or at least modular, analytical or simulative techniques can scale and provide models that can be effectively used; but after above a specific level of complexity, it is necessary to introduce approximations into models, to bypass the natural limits of the chosen technique (or, obviously, to switch to another type of analysis). This, however, can lead to less detailed system

Proceedings 28th European Conference on Modelling and Simulation ©ECMS Flaminio Squazzoni, Fabio Baronio, Claudia Archetti, Marco Castellani (Editors)
ISBN: 978-0-9564944-8-1 / ISBN: 978-0-9564944-9-8 (CD)

descriptions, and generally requires a complete redesign of the models. Of course approximations have to be carefully studied and justified, as their effects must not destroy the trustfulness of the model.

A possible approach to introduce approximations is to separately evaluate the effects of different layers or components of the architecture, to characterize their performances and find out if they can be somehow neglected or represented with simplified models with respect of the overall architecture.

In this sense, the goal of this paper is to support a wider modeling technique that aims to characterize generalized performance metrics of multicore CPU with a limited complexity. In this way, we can target massively distributed computing architectures, devoted to cloud and Big Data applications, designed to enable the description and the analysis of systems composed by a large number of independent components at different scale (see [1] [2] [3]). In this paper we separately explore the effects of multicore CPU and multithreaded applications, to understand if and how they should be significantly considered in modeling the target architectures.

In particular, we start by proposing characterization that exploits only exponential transition, for which both analytical and simulative techniques can be applied. Then we consider more complex models, which use fork and join of jobs and non-exponential durations, to better capture the relations between cores, threads and number of processes in parallel execution. Such models however, can only be solved via simulation, since analytical techniques would require a number of states that is too large to be considered.

III. RELATED WORKS

The problem represented by performance issues in systems based on multicore CPUs has been analyzed in literature by different points of view. To improve the performance, commercial architectural solutions have been designed and implemented like the Intel Multicore [4] or the AMD 16H [5] architectures. Moreover, several prototypes have been proposed and evaluated such as a utility-based mechanism that partitions a shared cache between multiple applications [6], or a cache-integrated network interface suitable for scalable multicores [7].

Even a proper evaluation of the benefits of these prototypes may be a challenging task due to the complex interaction between several factors. In the most of the literature instead of providing a complete characterization, just effects of one single component of the CPUs (or the system) have been investigated. For instance, [8] shows the different performances achieved when implicit or explicit cache management was used, whereas the authors in [8] propose a multiple linear regression model to investigate the impact of simultaneous multithreading on the performance. Also the effects of internal scheduling in memory was considered in [9] and [10]. Furthermore, the introduction of virtualization techniques, which can be modeled for instance by queuing networks as done in [11] and [12], can have a significant impact on the CPU performance [13]. As stated in [14] several factors introduced by virtualization affect the performance in way that are still not well-understood.

As the general approach is founded onto the definition or the application of benchmarks that are run on real systems to tune analytical or simulative models, in this paper in vivo measures will be used to validate the proposed models, to obtain a reliable base on which more general performance consideration can be carry out (as, e. g., in [15]), and try and get some general indications about the influence of multithreading and multicore on the overall performances of a complex system architecture.

IV. ANALYTIC RESULTS

We start by focusing on single threaded applications running in multicore environments. A classical queuing model for a multi-processor system running a closed workload, with N identical jobs that are characterized by exponentially distributed service times from CPU and I/O operations is shown in Fig. 1. In particular, the system can be modeled by two stations: one single server that represents the I/O component, and a multiple server that represents the CPU and the scheduler of the operating system. The multiplicity of the server of the queue corresponds to the number of cores of the CPU. As known, the behavior of the system strongly depends on

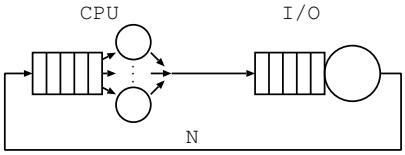

Fig. 1. CPU-IO Queueing Model.

the type of the workload, i.e.: I/O-bound or CPU-bound. When considering multicore CPU, the influence become even stronger. To properly study this effect, we start by focusing on applications having a total execution time T. We measure how much an application is I/O-bound as the fraction of time α the program spends doing I/O. In this way, if $\alpha = 0$, the workload is completely CPU-bound, and if $\alpha = 1$ the program does only I/O. In particular, we set in the queueing network of Fig. 1,the *demands*[1] of the I/O ($D_{I/O}$) and CPU (D_{CPU}) stations to:

$$D_{I/O} = \alpha T, \qquad D_{CPU} = (1-\alpha)T. \tag{1}$$

In Fig. 2a, the average system response time, as function of the number of jobs in the system N, is shown for a CPU-only application ($\alpha = 0$), running on a processor with 1, 2, 3 and 4 cores. As it can be seen, since the considered job is entirely CPU-bound, the mean response with c core follows a very simple relation that can be obtained in closed form by applying the Little's law:

$$R(N,c,T) = \begin{cases} T & \text{if } N \leq c \\ \frac{T \cdot N}{c} & \text{if } N > c \end{cases} \tag{2}$$

[1]With the term *demand* we characterize the total time spent by an application in one of its phases

The behavior has a very simple explanation: if $N \leq c$, there is at least a core for each job, so no queue is ever formed, and the response time of each job is equal to its demand. If $N > c$, the available cores need to be shared among the considered jobs. The total available capacity of the c cores is thus uniformly distributed among the jobs, leading to a response time $R = T \times N/c$. When I/O is present, ($\alpha > 0$), the CTMC underlying the model shown in Fig. 1 is isomorphic to the one corresponding to a M/M/c/N, where the arrival rate is equal to the inverse of the I/O demand, and the service rate corresponds to the inverse of the CPU demand. In particular, if we define $\lambda = \frac{1}{D_{I/O}}$, $\mu = \frac{1}{D_{CPU}}$, and we call π_0 the probability of having all the jobs doing I/O and π_N the probability of having all the jobs in the CPU, the mean response time can be computed as:

$$\pi_0 = \left(\sum_{k=0}^{N} \frac{\lambda^k}{\prod_{i=1}^{k} (\min(i,c)\mu)} \right)^{-1} \quad (3)$$

$$\pi_N = \pi_0 \frac{\lambda^N}{\prod_{i=1}^{N} (\min(i,c)\mu)} \quad (4)$$

$$X = \mu(1-\pi_N) \quad (5)$$

$$R = N/X \quad (6)$$

Eq. 6 is the Little's law, Eq. 5 is the Utilization law applied to the I/O (since $1-\pi_N$ is the utilization of the I/O), and the last two equations are the queuing lengths probability of a M/M/c/N queue (see for example [16]). When the percentage of I/O starts to increase (Fig. 2b, c and d), the effect of the multicore becomes less and less evident. In particular, when $\alpha = 0.24$, there is no more difference between using a 3 or a 4 cores CPU; for $\alpha = 0.36$, all the systems performs as dual-core; and for $\alpha > 0.48$, there seems to be no more advantage in using more than one core. The reason is that the bottleneck switches from the CPU to the I/O, thus canceling the effects improvements that the increased number of cores can bring to the main processor unit. This type of bottleneck switch is further emphasized in Fig. 3, where the effect of parameter α on response time is shown for a workload $N = 20$ when considering $c = 1 \ldots 4$ cores. In particular, when the system is single core, the behavior is perfectly symmetric, since the bottleneck switches exactly at $\alpha = 0.5$. Instead when the number of cores increases, the bottleneck switch point tends to reach $\alpha = 0$, that is: even small percentage of I/O can reduce the improvements provided by a higher number of cores. Let us call α^* the value of α at which the bottleneck switches from CPU to I/O. At α^*, both the CPU and the I/O have exactly the same demand:

$$\frac{D_{CPU}}{c} = D_{I/O} \quad (7)$$

By inserting in Equation 7 the definitions of the demands as function of the total service time T and α^*, we obtain:

$$(1-\alpha^*) \cdot T = c \cdot \alpha^* \cdot T$$

from which we can easily determine α^*:

$$\alpha^* = \frac{1}{c+1} \quad (8)$$

Summarizing, increasing the number of cores the fraction of CPU required for it to be the bottleneck, decreases hyperbolically. For example, with 8 cores, an I/O part that is around 12%, is enough to move the bottleneck away from the CPU, and thus reducings the benefits provided by more cores. Figure 3 shows the location of α^* determined by Equation 8 with a cross placed over the response time curve. As it can be seen, the response time has its minimum in all cases for $\alpha = \alpha^*$: that is the *common saturation point*, the point for which the demands of the I/O and of the CPU are equal. By inverting the definition of α^*, we can determine the best number of cores that can be used with an application, provided that we know its demand in term of CPU and I/O. In particular, the best number of cores c^* can be computed as:

$$c^* = \left\lceil \frac{D_{I/O}}{D_{CPU}} \right\rceil \quad (9)$$

With similar reasoning, we can compute the asymptotic

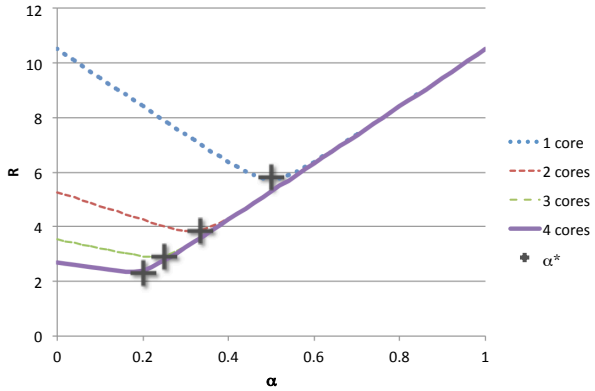

Fig. 3. Response time for $N = 20$ and different numbers of cores as function of α, together with the common saturation point α^*.

bounds R^* to which the system response time tends as function of α and the number of cores c. In particular we have:

$$R^*(N) \geq \max\left(N \cdot \frac{D_{CPU}}{c}, N \cdot D_{I/O}\right) = T \cdot \max\left(N \cdot \frac{1-\alpha}{c}, N \cdot \alpha\right) \quad (10)$$

V. FITTING CPU AND I/O DEMANDS IN VIRTUALIZED MULTICORE SYSTEMS

To see a practical application of the previous model, we use it to fit the response time of two components of the workload generated by a well known benchmark: the DaCapo suite [17]. In particular we use `sunflow` and `batik`: the former is a highly parallel application, which perform ray-trace rendering of 3D computer generated images. The latter produces Scalable Vector Graphics images, using the component Apache Batik: the benchmark is mainly single threaded, even if it can generate several threads during the transcoding of the images elements analyzed by the library. As seen in the analytical results of Section IV, the response time may be mostly influenced by the demand of the CPU or by the demand of the I/O, which in turn depend on the number of cores and

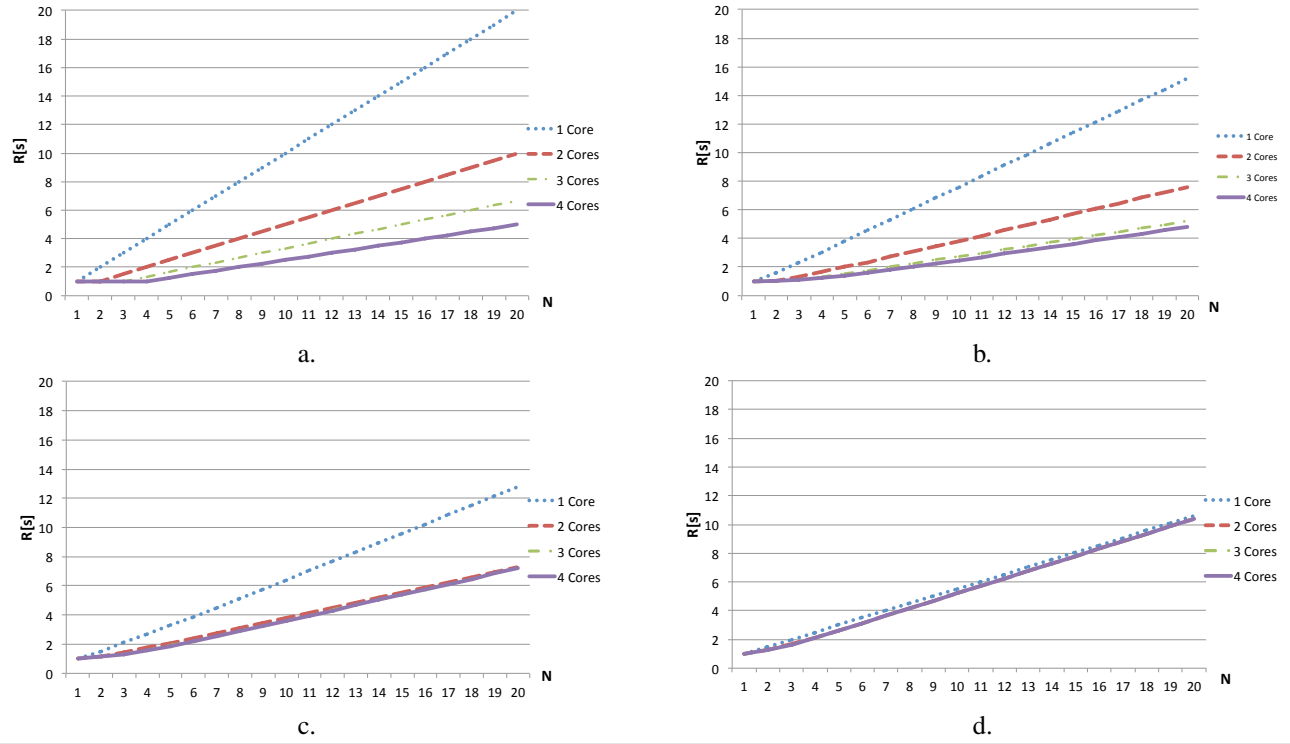

Fig. 2. CPU-IO switch. a) CPU=100%. b) CPU=76%. c) CPU=64%. d) CPU=52%.

on the fraction of I/O, respectively. We run the benchmarks on Amazon EC2 virtual machines running the Linux OS, and measure the I/O percentage of the applications using the *iostat* command. We then perform a fitting procedure to determine the CPU demand from the collected data using the *"GRG solver"* in Microsoft Excel, one of the most widely available tools suitable for the considered task. We fit the measures

Simple fit

Benchmark	Mean err.	Demand
sunflow	4,07%	15,524473
batik	17,42%	2,2455743

Load dependent fit

Benchmark	Mean err.	Demand
sunflow	0,85%	15,70436
batik	4,53%	2,2142329

Fig. 4. Fitting errors of the load-independent (upper) and of the load-dependent (lower) models.

of eight-cores VMs against the model presented in Figure 1. We estimate the CPU demands D_{CPU} and $D_{I/O}$ as the values that minimize the squared distance between the model and the measured response times. The fitting results are shown in the left column of Figure 4. As it can be seen, only the fitting with `sunflow` data produces accurate results, with a mean error of 4%. A visual comparison of the measured response time, and the one obtained with the model is given in Figure 5. The achieved accuracy strongly depends on the characteristics of the benchmark and of the cloud environment: `sunflow` performs 3D rendering, which is an highly parallel and memory consuming task. In this case the L2 shared cache is not exploited at its best, because it become saturated almost immediately. Since the M/M/c does not consider caching, the model is capable of describing `sunflow` behavior accurately. `batik` however performs sequential tasks only: this causes an average error of 17.4%, as it can be seen in Figure 6. In order

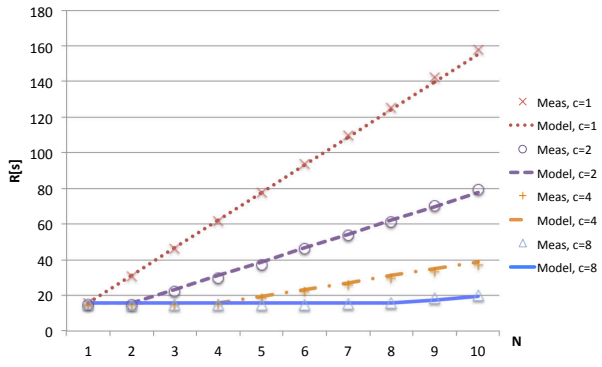

Fig. 5. Comparison of model results and measurements for `sunflow`.

to improve the results, we can use the approach proposed in [18]. In particular, we can consider a load-dependent CPU service station. We call $\lambda(c)$ and $\mu(n,c)$ the I/O and the CPU

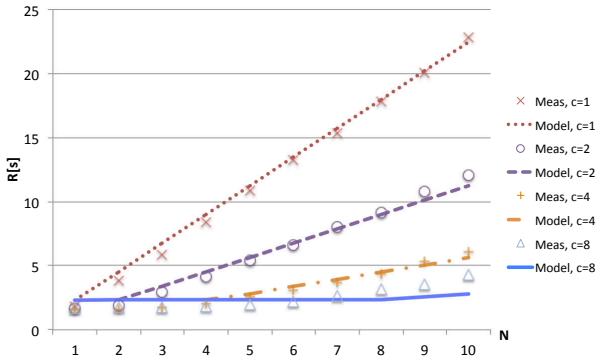

Fig. 6. Comparison of model results and measurements for `batik`.

service rates when there are c cores available and n jobs in the system. The response time of the proposed model can be computed using Equations 5 and 6, with $\pi_N(c)$ and $\pi_0(c)$ defined as follows:

$$\pi_N(c) = \pi_0(c)\frac{\lambda(c)^N}{\prod_{i=1}^{N}(\mu(\min(i,c),c))} \quad (11)$$

$$\pi_0(c) = \left(\sum_{k=0}^{N}\frac{\lambda(c)^k}{\prod_{i=1}^{k}(\mu(\min(i,c),c))}\right)^{-1} \quad (12)$$

We then repeat the parameter fitting procedure to estimate both $\lambda(c)$ and $\mu(n,c)$. In this way, the model can accommodate for the fluctuations due to the internal architecture of the CPU when the load is lower than the number of cores. The effective service rate remains constant when the load becomes greater than the available cores: from that point on, the CPU is fully utilized, and its behavior operates at a speed which can be considered constant. Results are shown in Figure 7, while errors are reported in the lower part of Figure 4. As it can be seen, the mean error rate reduces significantly for both benchmarks.

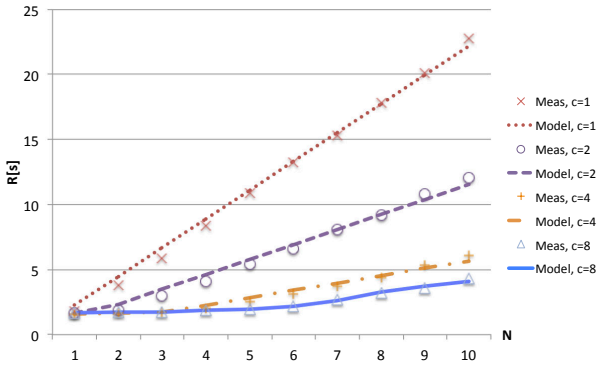

Fig. 7. Comparison of results for the load-dependent model and measurements for `batik`.

CONSIDERING MULTITHREADED APPLICATIONS

As shown in [19], multicore architectures are particularly efficient when running multithreaded applications: in this case, when the load is less than the total number of available cores, threads can be run in parallel to reduce the overall response time. For example, in Figure 8, the response time of the `sunflow` benchmark is shown for different number of threads and cores combinations. Since `sunflow` computes rendering of images, it is highly parallelizable: the picture can be segmented into independent parts that can be produced in parallel. As it can be seen, when the computation is split into several threads, the response time is reduced when the product between the number of jobs and the number of threads is less than the available number of cores. As soon as the number of jobs is greater than the number of cores, there is no difference between the response times independently on the number of threads.

Figure 9 shows instead the response time of the `batik` benchmark. Since `batik` is not parallel and it is composed mainly by serial tasks, there are no appreciable differences in the response times with respects to the number of threads. Instead there is a difference when considering a number of cores greater than the total number of jobs.

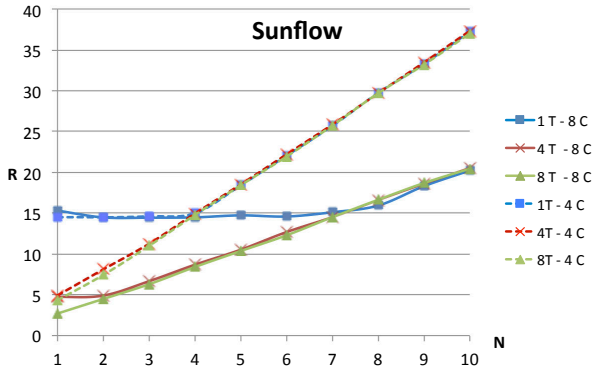

Fig. 8. `sunflow` mean response time in seconds under different configurations of cores/threads, increasing N.

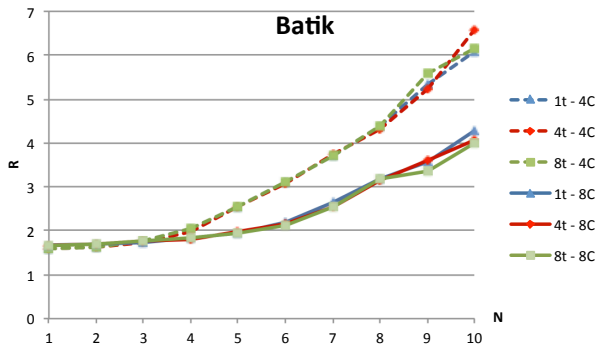

Fig. 9. `batik` mean response time in seconds under different configurations of cores/threads, increasing N.

The model presented in Figure 1 is not able to correctly characterize the parallelism due to the threads. We then propose an extended model, based on the simplifying assumption that a job executes both a serial and a parallel section, as

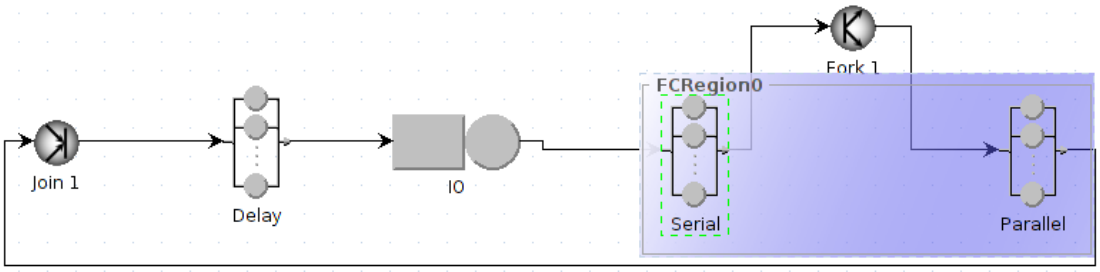

Fig. 10. Model for multithreaded applications in multicore environments.

shown in Figure 10. Each job starts executing the I/O phase, implemented by a standard queue, with a single servant and a First Come First Served discipline. Then, the job executes the serial part, and after that it forks into as many requests as the number of threads that composes the parallel part of the application. Each job is then served by another queue that represents the parallel part of the process. Finally, all the jobs generated by a fork are merged back into a single one using a join operation. The main difficulty is that the service of both the serial and the parallel part is the same: the two queues represents instead the same CPU. This is solved by introducing a *finite capacity region* (FCR), that is a constraint on the total population that is contained in a subset of the model. In particular, we fixed a constraint that tells that the total number of jobs inside the FCS is less or equal to the total number of cores. Then, to allow the parallelism of the cores, both the serial and parallel components of the CPU are modeled using infinite server semantics. Even if the model is very simple, the introduction of the fork and join and of the FCR, prevents it from being solved with analytical and numerical techniques (unless for very small value of the parameters). We thus resort to discrete event simulation: in particular we use the *jSim* component of the JMT - Java Modeling Tool [20]. The considered tool, requires a *reference station* to compute the performance indices: we have chosen to model this by explicitly adding an infinite-server station that represents the start-up of the benchmark (called Delay in Figure 10), characterized by a negligible service time. The model is thus completely characterized by six parameters. In particular, the application is defined through four parameters: the I/O duration, the duration of the serial part, the duration of the parallel part and the number of threads in the parallel part (that is, the number of tasks in which a job is split by the fork node). The number of cores that reflect the hardware architecture is modeled by the size of the FCR. Finally, the number of jobs in the system (for what concerns the serial and the I/O parts) is defined by the initial population of the network.

In this case, also the characterization of the parameters is a challenging task. As for the simpler model, we start by estimating the I/O duration using the I/O utilization values collected by `iostat` command. Then, from the CPU time we estimate the value of the demand of the serial and parallel part. In this case, we adopt different approaches according to the considered benchmark. Since `sunflow` is highly parallelizable we estimate the demands of the serial and parallel part using Amdahl's Law [21] and the measurements of the system when it is running one job and as many threads as cores. The results of the matching are given in Figure 11 for the four core case, and in Figure 12 for eight cores. The average relative error of the procedure is about 4.47%, similar the one obtained with the simple model of single threaded application considered in Figure 4.

However for `batik` such approach does not provide good results: in this case determining the demands from the measured response time with a single job in the queue can match either the left or the right part of the curve, depending on wether we give more importance to the system with just one thread, or to the system with as many threads as the number of cores. To obtain better results, we put the serial part (since the application is not parallelizable) dependent on the number of jobs in the system. We estimate the value using a basic fitting procedure that perform a gradient descent algorithm with a small number of iteration to cope with the complexity of the solution computed via simulation. Results are shown in Figure 13 for the four core case, and in Figure 14 for eight cores. In this case, the procedure is able to obtain an average error of 9.3%.

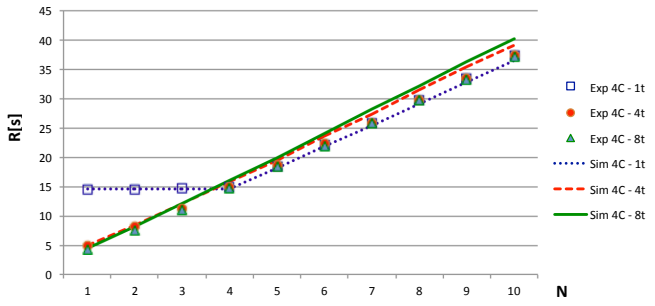

Fig. 11. `sunflow` mean response time in seconds for four cores, and different threads and N.

VI. CONCLUSIONS

In this paper we proposed two approaches to characterize multithreaded applications in multicore environments characterized by a limited number of parameters. Some insights

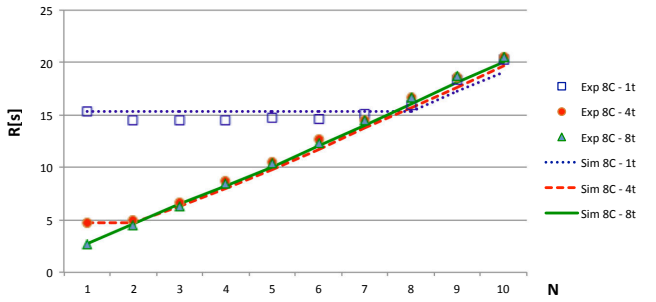

Fig. 12. `sunflow` mean response time in seconds for eight cores, and different threads and N.

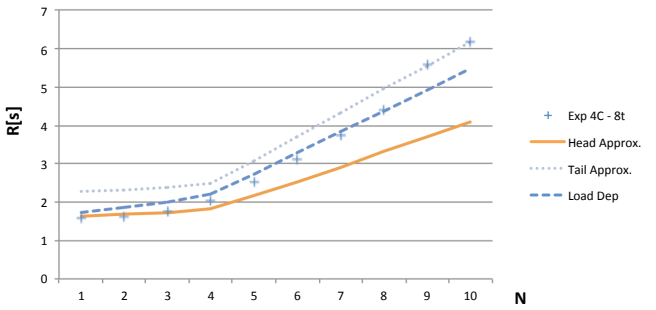

Fig. 13. `batik` mean response time in seconds for four cores, and different threads and N.

on how such parameters can be derived from measurements coming from executions of real applications on actual multicore machines have been given. The proposed results will be the foundation on which future works targeting Big Data application and cloud infrastructures will be based.

References

[1] A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri, "Exploiting mean field analysis to model performances of Big Data architectures," *Future Generation Computer Systems*, no. 0, pp. –, 2013.

[2] E. Barbierato, M. Gribaudo, and M. Iacono, "Performance evaluation of nosql big-data applications using multi-formalism models," *Future Generation Computer Systems*, vol. to appear, 2013.

[3] A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri, "Modeling performances of concurrent big data applications," *Software: Practice and Experience*, vol. to appear, no. 0, pp. –, 2014.

[4] J. Doweck, "Microarchitecture and smart memory access," 2006. [Online]. Available: http://software.intel.com/sites/default/files/m/3/4/d/6/3/18374-sma.pdf

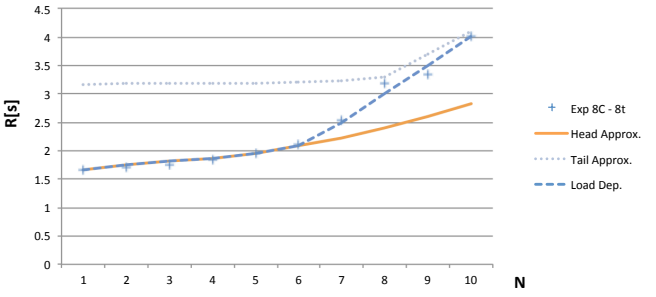

Fig. 14. `batik` mean response time in seconds for eight cores, and different threads and N.

[5] "Software optimization guide for amd family 16h processors," 2013. [Online]. Available: http://support.amd.com/TechDocs/52128_16h_Software_Opt_Guide.zip

[6] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 39. Washington, DC, USA: IEEE Computer Society, 2006, pp. 423–432.

[7] S. G. Kavadias, M. G. Katevenis, M. Zampetakis, and D. S. Nikolopoulos, "On-chip communication and synchronization mechanisms with cache-integrated network interfaces," in *Proceedings of the 7th ACM international conference on Computing frontiers*, ser. CF '10. New York, NY, USA: ACM, 2010, pp. 217–226.

[8] S. Schneider, J.-S. Yeom, and D. Nikolopoulos, "Programming multiprocessors with explicitly managed memory hierarchies," *Computer*, vol. 42, no. 12, pp. 28–34, Dec.

[9] F. Liu, X. Jiang, and Y. Solihin, "Understanding how off-chip memory bandwidth partitioning in chip multiprocessors affects system performance," in *HPCA 2010*, Jan., pp. 1–12.

[10] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *HPCA 2010*, Jan., pp. 1–12.

[11] D. A. Menasce', "Virtualization: Concepts, applications, and performance modeling," in *Proc. of The Computer Measurement Groups 2005 International Conference*, 2005.

[12] F. Benevenuto, C. Fernandes, M. Santos, V. A. F. Almeida, J. M. Almeida, G. J. Janakiraman, and J. R. Santos, "Performance models for virtualized applications." in *ISPA Workshops*, ser. Lecture Notes in Computer Science, G. Min, B. D. Martino, L. T. Yang, M. Guo, and G. Rnger, Eds., vol. 4331. Springer, 2006, pp. 427–439.

[13] L. Cherkasova and R. Gardner, "Measuring cpu overhead for i/o processing in the xen virtual machine monitor," in *Proc. of the USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 24–24.

[14] N. Huber, M. Von Quast, F. Brosig, and S. Kounev, "Analysis of the performance-influencing factors of virtualization platforms," in *Proceedings of the 2010 international conference on On the move to meaningful internet systems: Part II*, ser. OTM'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 811–828.

[15] M. Gribaudo, P. Piazzolla, and G. Serazzi, "Consolidation and replication of vms matching performance objectives," in *Analytical and Stochastic Modeling Techniques and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7314, pp. 106–120.

[16] L. Kleinrock, *Queueing Systems, Volume 1: Theory*. New York, NY: John Wiley & Sons, 1976.

[17] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann, "The dacapo benchmarks: Java benchmarking development and analysis," *SIGPLAN Not.*, vol. 41, no. 10, pp. 169–190, Oct. 2006. [Online]. Available: http://doi.acm.org/10.1145/1167515.1167488

[18] D. Cerotti, M. Gribaudo, P. Piazzolla, and G. Serazzi, "Flexible cpu provisioning in clouds: A new source of performance unpredictability," in *QEST*, 2012, pp. 230–237.

[19] D. Cerotti, P. Piazzolla, M. Gribaudo, and G. Serazzi, "End-to-end performance of multi-core systems in cloud environments," in *EPEW*, 2013, pp. 221–235.

[20] M. Bertoli, G. Casale, and G. Serazzi, "The jmt simulator for performance evaluation of non-product-form queueing networks," in *Proc. of the 40th Annual Simulation Symposium (ANSS)*, 2007, pp. 3–10.

[21] G. M. Amdahl, "Validity of the single-processor approach to achieving large scale computing capabilities," in *AFIPS Conference Proceedings*, vol. 30. AFIPS Press, 1967, pp. 483–485.