A Review on Task Performance Prediction in Multi-core Based Systems

Frédéric Pinel, Johnatan E. Pecero, Pascal Bouvry
Computer Science and Communications Research Unit
University of Luxembourg
L-1359, Luxembourg
Email: firstname.lastname@uni.lu

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

***Abstract*—Operators of data centers are faced with the challenging goal of hosting applications that meet agreed service levels, at minimal operating costs. A significant part of these costs is energy related. Successfully reaching this goal requires optimal task-to-machine assignments. This activity relies on accurate energy and performance prediction. Widespread use of multi-core, multi-processor machines complicate past prediction methods. Therefore, this paper suggests to revisit task profiling, a method based on observations of actual task execution. As a first step in this direction, this paper reviews methods for task profiling, which account for the contention present in multi-core processors.**

***Keywords*-scalable architectures; task profiling; performance of system; energy-efficiency;**

I. Task performance prediction

This paper is a review of past efforts to predict a task's performance, on a multi-core, multi-processor based machine. A *task* is a component of a software *service*, which performs some meaningful transaction. The tasks of a service can be distributed across machines. This increases the benefits of accurate task performance prediction, by enabling consolidation, for operators of clusters, data centers, grids or clouds. However, the approach could also be used for a single machine, where the various tasks composing the service run concurrently on different cores.

A. Motivation

Performance prediction contributes to the problem of efficiently running a software service which demands a set quality of service (QoS), as defined in a service level agreement (SLA). The business owners of the service place constraints on the performance of the service, typically defining objectives for latency (response time) or throughput. It is therefore necessary to operate a computer system with the business objectives in mind. Owners of a computer based solution care little about the individual performance of the various technical components.

Successfully addressing this problem is of broad interest. Given the complexity of operating a data center, more businesses rely on third parties to host their service. Cloud computing is the latest instance of this trend, its rapid adoption highlights the importance of this question. It is of interest to such computing providers to satisfy SLA while minimizing their costs, such as infrastructure and energy.

Although, the focus of this paper is the problem of independent task mapping (placing the tasks onto the different machines, or processors), several key activities involving distributed systems depend on the accurate prediction of task performance. These include runtime activities of a distributed system, such as:

- task scheduling on the available computing resources,
- load balancing on the same infrastructures.

Several design time activities may also benefit from an accurate prediction model to guide their decision process:

- capacity planning of a distributed infrastructure, such as data centers, web hosting centers, cloud service providers,
- programming, because programming for performance requires a model for task performance.

B. Runtime task profiling approach

To make correct decisions, the mapper in a distributed system needs to evaluate the different alternatives. This implies some *prediction model* of the performance of the tasks, on the distributed system available.

From the performance predictions of the individual tasks, the mapper is able to evaluate the overall performance of the service. This exposes a key difference between operating system scheduler(s) and the higher level mapper considered here. The former cannot assess the overall service performance because their scope is the machine, and more generally because their objective is resource management (of hardware), rather than business performance. This limitation is also present in multi-server (micro-kernel) operating systems [1]. This can justify the use of such a predictor for single machines.

This is not a new topic; however the widespread use of multi-processor, multi-core based machines and their system programs prompt a review of past models.

Models to predict performance were usually developed from a detailed understanding of the inner workings of tasks and machines [2]. However, the growing complexity of these

multi-core based computers do not lend themselves well to such an approach. These machines although sometimes considered parallel, actually share several components (memory, last level caches, I/O interfaces), which lead to contention. This contention considerably impacts the performance of a task [3], [4], and [5]. Therefore, the actual performance of a task depends on the concurrent activity on other cores and processors.

Operating systems (virtual memory management, time-sharing) and related tools (power management) further add to this hardware complexity. For example, paging faults considerably hurt an expected performance [6]. Tasks are not simpler. They are often considered defined by the source code in a high level language, although their behavior is set by compilers (optimization techniques) and runtime libraries. Moreover, some tasks can change behavior at execution time, in order to use a different amount of resource. For example: a task that relies on the slab allocator [7] can be requested to reduce its memory footprint. This prevents accurate prediction of a task's performance based on past performance in a different environment (different loads from different concurrent tasks).

All these components prevent the accurate modeling of the computation, and the performance prediction of a given task on a given machine. Even if such an accurate model was developed, it would only be valid up to the next change in any of its parts: hardware, operating system, compiler, runtime library or task source program.

The question then is how to model task performance, when it depends on so many parts, each difficult to accurately model. Preferably, the prediction model should work for all possible machines and tasks. A possible approach is to avoid, as much as possible, a priori knowledge on the inner workings of the system, but should rather observe the execution of the tasks on the actual machines (hardware and related software such as operating system). This is sometimes called runtime task profiling. The goal is not a descriptive model, but a predictive one.

The suggested approach is based on a preliminary step to the mapping activity, which measures the execution time of the tasks, on the targeted system, but over a limited time only.

In addition, the task profiling should explore the effects of contention. This could be achieved by observing the task's performance when the machine is under various resource-specific loads, for a given profile run. Ideally, this extension should provide more accurate results but it ought not significantly increase the effort to build the model.

The next section presents a review of previous efforts in this direction.

II. Literature review

The past efforts are presented according to the field of application.

A. Distributed computing

As one of the first field to operate large computer infrastructures, high performance computing (HPC) has identified the need for task performance prediction a long time ago. However, the original works focused around proprietary parallel machines, along with dedicated system software.

Since then, large infrastructure shifted to distributed systems composed of commodity machines (PC), typically single processor. Recently, multi-processor, multi-core architectures, and upcoming many-core machines introduced a mixture of both.

The scheduling problem is clearly presented in [8], and [9], they also expose the need for:

- profiling the tasks to execute,
- benchmarking the machines.

However it does not take into account the sources of contention in recent multi-processor, multi-core computers.

Several works [10], [11], and [12] investigate task runtime prediction from their past performance, on the same machines. This approach derives from their context of application: a shared grid, where tasks controlled elsewhere are running on the same grid. This is a different problem from the one presented in this review. Here, the environment is completely controlled, but the problem is to map tasks in order to achieve specific results. Although this may seem an easier problem to solve, the problem of optimally mapping independent tasks on an heterogeneous system is NP-hard.

Ref. [13] presents a real-time advisor (RTA) to predict a task's performance. The RTA is used with a scheduler (real-time scheduler advisor), to place task on appropriate machines. The RTA predicts performance based on the observed runtime of the task when run on a vacant machine, and on the load of the machine where the task is planned to run. This formulation is due to the context for the prediction: how to map tasks on unreserved machines; which are under some load, outside of the mapper's control.

Different aspects of the problem are similar to the problem defined in this paper. Notably, the prediction of a task's runtime given a machine's load addresses the contention of shared resources. However, there are several differences. The tasks considered are computation-bound (busy loops), and therefore the shared resource is the processor. The machines are also single-core single-processor, so contention related to multi-core, multi-processor machines are not considered.

Moreover, the predictor relies on the task's measured runtime, on the machine available, but only when run on a vacant machine (loadless). Finally, the scheduling

problem is slightly different, the objective in [13] is to select the most suitable machine for a task, while the mapping problem defined in Section I is combinatorial: it tries to find the optimal mapping of all tasks, onto the vacant machines.

Ref. [14] designs a prediction model for the grid, but opt to focus on a specific application in order to improve its accuracy. Applications run on a grid, where each computing node is a cluster. This is different from the context of this paper, where a computing node is a machine. Contention at the machine level is therefore not included. Their approach is based on a limited run of the applications: executing the application against some input data, on one cluster. Predictions of runtime in other configurations (different data sets or clusters) are extrapolated from this initial measurement. Restraining the scope to a specific application is not part of the problem defined in Section I, and the service considered is different from scientific or data mining applications.

Ref. [15] addresses a similar problem and suggest a similar approach. The host machine used for their study is a 32 processor SMP machine, but their work is applicable to other configurations. However, the tasks are quite specific. The execution model is that of work-stealing. A task is executed by the first available processor. It is not a long-running service, but rather a job, which is started and ends. A task can create other tasks. The objective of the study is to predict performance of the overall application. The tasks are profiled by monitoring their execution over a limited time. Memory contention is partially accounted for. The number of threads used for each task execution is varied, in order to assess the scalability of each task. A task which does not scale well reflects some contention (locking, cache conflicts, etc) within the task.

However, the objective is the task's scalability, and not the characterization of each task: how it can cause contention with any other task.

The problem of application performance prediction is studied in [16]. The authors propose an exploratory phase, called the pro-active training phase, which consists of running the tasks on the machines. They identify the cost of this training phase as a problem, and develop a method to minimize this effort. The method suggests to run the application on all nodes for a subset of the input values, and on the fastest machine for the full set of inputs. This formulation assumes a HPC application, which is run over a range of inputs. The data collected is then used for prediction of the real application on various machines.

The problem formulation and the HPC setting differ from the problem defined here, but the approach to rely on actual executions and their observed runtime is very similar. The effort to minimize the preliminary phase is noteworthy, even if the solution is HPC specific. Moreover the question of contention in multi-core based machines is not considered.

Ref. [17] predicts the runtime of a task on a machine from the load of the task in isolation, the characteristic of the machine and the current load of the machine (due to other tasks). This prediction also relies on 5 rules that capture the interaction of tasks.

This recent contribution is interesting because it bases its prediction on actual task execution, and limits their cost by only executing tasks in isolation. It also casts the problem in the context of quality of service, which is the end-user perspective included in the proposed direction. The literature review is also noteworthy.

However, there are differences between approaches. The model used to predict task load does not address contention in multi-core, but mentions it as an operating system concern. The tasks considered are essentially CPU-bound, because this allows the authors to link host load to task runtime. While this may be true for CPU-bound tasks, it is most unlikely in memory, network or any other contention prone situation. The domain of application for the profiling is HPC. Therefore the tasks are considered long-running (measurements are based on 5 second sampling), and directly related to a user id on the machine. The tasks in the present paper are different from these HPC tasks, because they are the instructions necessary to process a request, as part of a daemon or service. The 5 rules necessary to predict runtimes from exploratory data are based on an understanding of the machines and the nature of task execution, which represents a big assumption.

Ref. [18] introduces their performance predictor: Dimemas. This simulator relies on execution traces of applications, on some characterized hardware. It can then predict the runtime of the same application on different hardware. The CPU burst, and network activity is considered in the model. The context is different from the one presented here, because the tasks are MPI-based HPC applications, and the varying hardware environment is the network performance. Therefore, a specific model for the task is defined, which is not the case in the proposed setting.

Ref. [19] classifies execution time prediction into three groups: code analysis, code profiling/analytic benchmarking, and statistical methods. From such classification, the authors present a hybrid approach: statistical and analytic benchmarking. This classification clearly exposes different approaches to task performance prediction. However, two hypothesis in this work define a different problem to the one studied here. First, each task is assumed to have exclusive use of the machine on which it runs, such that a task's exe-

cution time does not depend on other tasks. This is clearly different from the problem defined here, where contention for shared resource, by concurrently running tasks, is one key hypothesis. Second, this previous work considers that a task execution time depends on its input data. This is perhaps specific to HPC environments. However, there is no such hypothesis in this paper.

B. Thread scheduling

The method of profiling tasks based on their actual execution is proposed in [20]. This work looks at the execution of multiple threads of a process. It aims to identify data dependencies between threads. Data dependency occurs when multiple threads access the same data. Although the method is based on runtime analysis, compared to static source code analysis, it does not include machine characteristics, and does not consider contention beyond the data that threads share.

Ref. [21] characterizes the relation between workload and a machine's resource utilization (such as memory, CPU). The target application is capacity planning. The method relies on measured execution (called automated profiling). However, the relation sought does not involve performance estimation, because of the intended application in capacity planning. It does not consider contention in multi-core based machines.

C. Scheduling for simultaneous multi-threading architectures

The following papers consider how schedulers can improve the performance of threads when executed on simultaneous multi-threading (SMT) architectures. SMT improves instruction-level parallelism (ILP) by executing different threads at each cycle. The consequence is that some threads will achieve greater parallelism when co-scheduled together than other combinations.

This question bears some similarity with the question presented here, in Section I. Contention is present in SMT; however it is possible to minimize it in order to achieve greater performance. This depends on the nature of the threads (which are called tasks here). Co-scheduling threads on an SMT processor is analogous to mapping tasks on the same multi-core processor (or multi-processor machine). However, there are differences which prevent a direct application of the results from this field. The contention in SMT is limited in scope (processor core), while this paper places no restriction on the sources of contention within a machine (hardware and software stack). In addition, this paper is investigating a predictor for task runtime, while a SMT scheduler is concerned about processor utilization, a lower level information. Nevertheless, some methods from the field of co-scheduling for SMT could be applied to the question presented here.

Ref. [22], and [23] introduce a SMT scheduler which minimizes contention on a superscalar processor, to improve utilization and performance of the threads. Their scheduler initially co-schedules threads according to fair policy, and then attempts to discover which threads run well together, by deliberately changing the co-schedule and observing the resulting performance. The adaptive nature of the approach is unsuitable to the problem defined in Section I, because the unsuccessful co-schedules would impact the QoS and fail the SLA. Moreover, the combinatorial space of co-schedules is so large so as to make the above risk likely (because the scheduling is not limited to a SMT processor, but to an entire distributed system).

Ref. [24] targets the Simultaneous Multi-Threaded cores platform, such as Intel's HyperThreading. The intention of the study is similar to the approach suggested in Section I. The objective is thread scheduling to reduce contention. This scheduling can either be performed at the CPU level, or at the operating system level. The model is based on measurements of a real thread execution. However, there are notable differences. The approach is not exploration based (it does not require a preparation step which explores the behavior of tasks), but adaptive. The measurements deal with resource usage, such as caches and registers. The approach proposed in Section I measures overall program performance, as perceived by the users. Their model is based on the detail knowledge of the internals of the SMT processor, and therefore uses of a simulator to obtain results. The approach presented here tries to avoid both this knowledge and the use of a simulator.

D. Real-time and embedded system scheduling

Ref. [25] proposes an energy-efficient soft real-time scheduler. The scheduler relies on runtime predictions, based on limited task execution, which is the approach considered in the present review. Soft real-time (meeting a fraction of all deadlines) expresses the problem of meeting SLA requirements, because SLA typically allows some deadlines to be missed, and sets penalties when more deadlines are missed. The deadlines capture the end-user view of performance. However, contention is not part of the model (which focuses on the CPU, for specific tasks). Also, energy efficiency is a consequence of DVFS control by the scheduler, which is not necessarily better than the race-to-idle policy. Finally, this scheduler is the finest grain operating system scheduler, which operates at a lower level than this paper selected.

Ref. [26] proposes a real-time scheduler where activities are subject to resource constraints. The constraint is that shared resources are accessed sequentially. They mention both physical resources, such as disks, and logical resources, such as critical sections guarded by mutexes. Only one

task (according to our definition, not theirs) can access a shared resource at a time. Tasks are defined with statistical properties. However, how such properties are obtained is not described; it is likely that they are derived from actual executions. However, the task properties do not include resource contention. Constraints on shared resources are managed through scheduling access to resources, under a given model. Although their study does not match this review's scope of task profiling, it does address the higher level question of scheduling tasks under resource contention.

E. Internet Protocol routers

Some closely related works from a different field than data center computing are presented in [27], [28], and [29]. These papers present and study programmable internet routers, based on network processors. One of the main issue with programmable routers is the mapping of tasks to processors.

Ref. [27] examines the suitability to internet routing of different machine architectures, They do mention contention as a critical bottleneck in network processors, but it is not part of the profiling or mapping study.

Dynamic profiling to support task mapping is proposed in [28], and [29]. This profiling aims to characterize the tasks. The dynamic profiling is motivated by the variability of the input traffic, both in volumes and nature. Contention is not part of the study, although it is presented in their previous paper.

III. Conclusion and future work

The problem of accurate performance prediction on multi-core, multi-processor machines based on a preliminary observation of a task's execution, has been little explored in the past. This is unfortunate because of the impact for data center operators, notably cloud service providers, which otherwise need to dedicate machines to run critical services, in order to avoid failing SLA.

Energy minimization is indirectly part of the scope, because it is related to number of machines and their load. However, it deserves to be more directly addressed because of the growing cost item it represents.

This review highlights the benefits of a mapper of a higher level than schedulers of operating systems. Indeed, such schedulers manage contention while preserving the current programmer's abstract view of a dedicated machine (memory, processor, I/O). An alternative, which increases the difficulty for the programmer is to account for varying resource availability inside the programs, and install a richer communication between schedulers and processes.

Future work involves building such a prediction tool, and evaluating the accuracy and the extra effort the preliminary profiling step implies. The next steps include the investigation of cloud-scale mappers, which rely on the aforementioned predictors.

Acknowledgment

This work is supported by the Fonds National de la Recherche Luxembourg, CORE Project GreenIT.

References

[1] N. H. Walfield and M. Brinkmann, "A critique of the gnu hurd multi-server operating system," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 30–39, July 2007. [Online]. Available: http://doi.acm.org/10.1145/1278901.1278907

[2] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox, "Pace: A toolset for the performance prediction of parallel and distributed systems," *International Journal of High Performance Computing Applications*, vol. 14, no. 3, pp. 228–251, Fall 2000.

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

[4] 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.

[5] 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.

[6] P.-H. Kamp, "You're doing it wrong," *Communications of the ACM*, vol. 53, no. 7, pp. 55 – 59, 2010. [Online]. Available: http://cacm.acm.org/magazines/2010/7/95061-youre-doing-it-wrong/fulltext

[7] J. Bonwick, "The slab allocator: an object-caching kernel memory allocator," in *Proceedings of the USENIX Summer 1994 Technical Conference on USENIX Summer 1994 Technical Conference - Volume 1*, ser. USTC'94. Berkeley, CA, USA: USENIX Association, 1994, pp. 6–6. [Online]. Available: http://portal.acm.org/citation.cfm?id=1267257.1267263

[8] T. Braun, H. Siegel, and A. Maciejewski, "Heterogeneous computing: Goals, methods, and open problems," in *High Performance Computing HiPC 2001*, ser. Lecture Notes in Computer Science, B. Monien, V. Prasanna, and S. Vajapeyam, Eds. Springer Berlin / Heidelberg, 2001, vol. 2228, pp. 307–318, 10.1007/3-540-45307-5_27.

[9] S. Shivle, P. Sugavanam, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, A. Kutruff, P. Penumarthy, P. Pichumani, P. Satyasekaran, D. Sendek, J. Smith, J. Sousa, J. Sridharan, and J. Velazco, "Mapping subtasks with multiple versions on an ad hoc grid," *Parallel Comput.*, vol. 31, pp. 671–690, July 2005.

[10] M. Dobber, R. D. van der Mei, and G. Koole, "Effective prediction of job processing times in a large-scale grid environment," in *HPDC*. IEEE, 2006, pp. 359–360.

[11] R. Wolski, "Experiences with predicting resource performance on-line in computational grid settings," *SIGMETRICS Performance Evaluation Review*, vol. 30, no. 4, pp. 41–49, 2003.

[12] P. A. Dinda and D. R. O'Hallaron, "Host load prediction using linear models," *Cluster Computing*, vol. 3, no. 4, pp. 265–280, 2000.

[13] P. A. Dinda, "Online prediction of the running time of tasks," *Cluster Computing*, vol. 5, pp. 225–236, 2002, 10.1023/A:1015634802585. [Online]. Available: http://dx.doi.org/10.1023/A:1015634802585

[14] L. Glimcher and G. Agrawal, "A performance prediction framework for grid-based data mining applications," in *IPDPS*. IEEE, 2007, pp. 1–10.

[15] R. Hoffmann and T. Rauber, "Profiling of task-based applications on shared memory machines: Scalability and bottlenecks," in *Euro-Par 2007 Parallel Processing*, ser. Lecture Notes in Computer Science, A.-M. Kermarrec, L. Bougé, and T. Priol, Eds. Springer Berlin / Heidelberg, 2007, vol. 4641, pp. 118–128, 10.1007/978-3-540-74466-5_14.

[16] F. Nadeem, M. Yousaf, R. Prodan, and T. Fahringer, "Soft benchmarks-based application performance prediction using a minimum training set," in *e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference on*, dec. 2006, p. 71.

[17] S. Seneviratne and D. C. Levy, "Task profiling model for load profile prediction," *Future Generation Computer Systems*, vol. 27, no. 3, pp. 245 – 255, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/B6V06-511TN6Y-3/2/47447581cb2d7ac8595db2a4d5792f20

[18] R. M. Badia, F. Escale, E. Gabriel, J. Gimenez, R. Keller, J. Labarta, and M. S. Muller, "Performance prediction in a grid environment," in *Grid Computing*, ser. Lecture Notes in Computer Science, F. Fernandez Rivera, M. Bubak, A. Gomez Tato, and R. Doallo, Eds. Springer Berlin / Heidelberg, 2004, vol. 2970, pp. 257–264, 10.1007/978-3-540-24689-3_32. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24689-3_32

[19] M. Iverson, F. Ozguner, and L. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *Computers, IEEE Transactions on*, vol. 48, no. 12, pp. 1374 –1379, dec 1999.

[20] C. von Praun, R. Bordawekar, and C. Cascaval, "Modeling optimistic concurrency using quantitative dependence analysis," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, ser. PPoPP '08. New York, NY, USA: ACM, 2008, pp. 185–196.

[21] T. Abdelzaher, "An automated profiling subsystem for qos-aware services," in *In IEEE Real-Time Technology and Applications Symposium*, 2000, pp. 208–217.

[22] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous multithreading processor," in *ASPLOS*, 2000, pp. 234–244.

[23] A. Snavely, D. M. Tullsen, and G. M. Voelker, "Symbiotic jobscheduling with priorities for a simultaneous multithreading processor," in *SIGMETRICS*. ACM, 2002, pp. 66–76.

[24] A. El-Moursy, R. Garg, D. H. Albonesi, and S. Dwarkadas, "Compatible phase co-scheduling on a cmp of multi-threaded processors," in *IPDPS*. IEEE, 2006.

[25] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," in *SOSP*, M. L. Scott and L. L. Peterson, Eds. ACM, 2003, pp. 149–163.

[26] H. Wu, B. Ravindran, E. D. Jensen, and P. Li, "Energy-efficient, utility accrual scheduling under resource constraints for mobile embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 5, pp. 513–542, August 2006.

[27] T. Wolf, "Challenges and applications for network-processor-based programmable routers," in *Sarnoff Symposium, 2006 IEEE*, march 2006, pp. 1 –4.

[28] X. Huang and T. Wolf, "Evaluating dynamic task mapping in network processor runtime systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, pp. 1086–1098, 2008.

[29] Q. Wu and T. Wolf, "On runtime management in multi-core packet processing systems," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '08. New York, NY, USA: ACM, 2008, pp. 69–78.