A Comparative Review of Contention-Aware Scheduling Algorithms to Avoid Contention in Multicore Systems

Genti Daci, Megi Tartari

Abstract. Contention for shared resources on multicore processors is an emerging issue of great concern, as it affects directly performance of multicore CPU systems. In this regard, Contention-Aware scheduling algorithms provide a convenient and promising solution, aiming to reduce contention, by applying different thread migration policies to the CPU cores. The main problem faced by latest research when applying these schedulers in different multicore systems, was a significant variation of performance achieved on different system architectures. We aim to review and discuss the main reasons of such variance arguing that most of the scheduling solutions were designed based on the assumption that the underlying system was UMA (Uniform Memory Access latency, single memory controller), but modern multicore systems are NUMA (Non Uniform Memory Access latencies, multiple memory controllers). This paper focuses on reviewing the challenges on solving the contention problem for both types of system architectures. In this paper, we also provide a comparative evaluation of the solutions applicable to UMA systems which are the most extensively studied today, discussing their features, strengths and weaknesses. For addressing performance variations, we will review Vector Balancing, OBS-X and DIO scheduling for UMA systems. While for NUMA systems, we will compare and discuss DINO and AMPS Schedulers which supports NUMA architectures aiming to resolve performance issues and also introduce the problems they have. This paper aims to propose further improvements to these algorithms aiming to solve more efficiently the contention problem, considering that performance-asymmetric architectures may provide a cost-effective solution.

Keywords: Uniform Memory Access(UMA), Multicore CPU systems, Contention-Aware Scheduling, Non Uniform Memory Access(NUMA), Vector Balancing Scheduling, OBS-X Scheduler, DIO Scheduler, DINO Scheduler, AMPS Scheduler.

1 Introduction

Contention for shared resources in multicore processors is a well-known problem. The importance of handling this problem is related with the fact that multicore processors are becoming so prevalent in desktops and also servers, that may be considered a standard for modern computer systems and also with the fact that this problem causes performance degradation. Let's consider a typical multicore system

S. Markovski, M. Gusev (Editors): ICT Innovations 2012, Web Proceedings, ISSN 1857-7288

described schematically in Figure 1, where cores share parts of memory hierarchy, that we call "memory domains", and compete for resources like last level cache (LLC), memory controllers, memory bus and prefetching hardware.

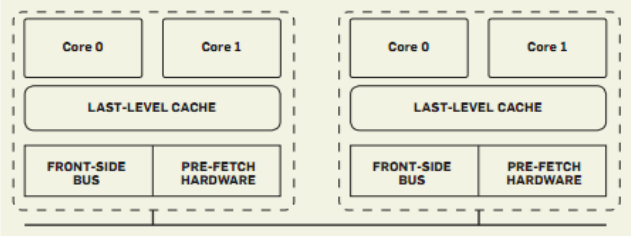

Fig. 1. A schematic view of a multicore system with 2 memory domains

Preliminary studies considered cache contention [5] as the most crucial factor responsible for performance degradation. Driven by this assumption, they focused on finding mechanisms to reduce cache contention like Utility Page Partitioning [8] and Page Coloring [7]. Successive studies [1] calculated the contribution that each of the shared resources in multicore processors have in degrading performance of such systems, concluding that contention for last level cache (LLC) was not the dominant factor in degrading performance. Based on this new conclusion, recent studies selected scheduling as an attractive tool, as it does not require extra hardware and it is relatively easy to integrate into the system. Contention-Aware scheduling [1][2][3][4] is proposed as a promising solution to this problem, because it reduces contention, by applying different thread migration policies. The major part of these studies, found solutions that could be applied only in UMA (Uniform Memory Access) systems, that are not suitable for NUMA (Non Uniform Memory Access). So for UMA systems we will discuss DIO Scheduler, that uses thread classification schemes like SDC [9], LLC miss rate [2], Pain metric [2], Animal Classes [10] to take the best scheduling decision; OBS-X scheduling policy based on the data provided by the OS dynamic observation of tasks behavior; Vector Balancing scheduling policy, that reduces contention for shared resources by migrating tasks based on the task activity vector information, that characterizes tasks regarding resource usage. For NUMA architecture, that still requires further research, is proposed DINO Scheduler. We will also discuss AMPS scheduler design for asymmetric-architecture multicore systems, that supports NUMA.

The rest of the paper is organized as follows: In Section 2 we argument why contention-aware algorithm is considered a promising solution to mitigating contention. In Section 3 we review and discuss the scheduling algorithms valid for UMA systems. In Section 4 we review and discuss the scheduling solutions proposed for NUMA architectures and we conclude in Section 5.

2 Contention-Aware Scheduling a Promising Solution

Preliminary studies on improving thread performance in multicore systems were mainly focused on the problem of contention for the shared cache. Cache partitioning has a significant influence on performance closely relating with execution time. J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan [5], implemented an efficient layer for cache partitioning and sharing in the operating system through virtual-physical address mapping. Their experiments showed a considerable increase of performance up to 47 %, in the major part of selected workloads. A number of cache partitioning methods have been proposed with performance objectives [7] [8] [25]. A. Fedorova, M. I. Seltzer, M. D. Smith [17] designed a cache-aware scheduler that compensates threads that were hurt by cache contention by giving them extra CPU time.

The difficulty faced from S. Zhuravlev, S. Blagodurov and A. Fedorova [2] in evaluating the contribution that each factor has on performance was that all the degradation factors work in conjunction with each other in complicated and practically inseparable ways.

To take into consideration the result of their work, it is proposed Contention-Aware Scheduling, that separates competing threads onto separate memory hierarchy domains to eliminate resource sharing and, as a consequence to mitigate contention. To design a contention-aware scheduler, initially we must choose a thread classification scheme, that predicts how they will affect each other when they will compete for shared resources and a scheduling policy, which assigns threads to cores given their classification. So the classification scheme serves to identify applications that must be co-scheduled or not. S. Zhuravlev, S. Blagodurov and A. Fedorova [2] help us with their contribution in analyzing the effectiveness of different classification schemes like:

- SDC (Stack Distance Competition), a well known method [9] for predicting the effects of cache contention among threads, based on the data provided from stack distance profiles, that inform us on the rate of memory reuse of the applications.
- Animal Classes is based on the animalistic classification of application introduced by Y. Xie and G. Loh [10]. It allows classifying applications in terms of their influence on each other when co-scheduled in the same shared cache.
- Miss Rate is considered as the heuristic for contention, because it gives information for all the shared resources contention.
- Pain Metric is based on *cache sensitivity* and *cache intensity*, where sensitivity is a measure of how much an application will suffer when cache space is taken away from it due to contention; intensity is a measure of how much an application will hurt others by taking away their space in a shared cache.

The results of evaluation of effectiveness of these classification schemes, show that the best contention predictor is Miss Rate [2][11]. A high miss rate exacerbates the contention for all of these resources, since a high-miss-rate application will issue a large number of requests to a memory controller and the memory bus, and will also be typically characterized by a large number of prefetch requests, while SDC performed

worse because it does not take into account miss rates in its stack distance competition model and it models the performance effects of cache contention, which is not the only cause of degradation.

As a perfect scheduling policy, it is used an algorithm proposed by Y. Jiang, X. Shen, J. Chen, R. Tripathi [16].This algorithm is guaranteed to find an optimal scheduling assignment, i.e., the mapping of threads to cores, on a machine with several clusters of cores sharing a cache as long as the co-run degradations for applications are known. Jiang's methodology uses the co-run degradations to construct a graph theoretic representation of the problem. The optimal scheduling assignment can be found by solving a min-weight perfect matching-problem.

3 Proposed Schedulers for UMA Systems

Several studies investigated ways of reducing resource contention and as mentioned above in Section2, one of the promising approaches that emerged recently is contention-aware scheduling [2][3][4]. This represents a promising solution, as several research co-scheduled tasks based on memory bandwidth or other shared resources. We mention here the co-scheduling tasks proposed for SMP [19,22] and for SMT [20]. These studies of contention-aware algorithms were focused primarily on UMA (Uniform Access Memory) systems, where there are multiple shared LLCs, but only a single memory node equipped with a single memory controller, and memory can be accessed with the same latency for any core. In this section we will review and evaluate OBS-X, Vector Balancing scheduling policy, and DIO scheduler by discussing their features, merits, but also their gaps.

3.1 OBS-X Scheduling Policy based on OS Dynamic Observations

According to R. Knauerhase, P. Brett, B. Hohlt, and S. Hahn [3], in a multicore environment the Operating System (OS) can and should make observations of the behavior of threads running in the system. These observations, combined with knowledge of the processor architecture, allow the implementation of different scheduling policies in the Operating System. Good policies can improve the overall performance of the system or performance of the application.

The performed experiments on this study have included various software and hardware environments. The lack of intelligent thread migration and also the fact that OS handles cores as independent, without taking into account that they share resources represent the challenges faced by R. Knauerhase, P. Brett, B. Hohlt, and S. Hahn during this study, where they found a policy to address these challenges, as the traditional operating system scheduler does not take into account the fact that amount of contention is quite dynamic because it depends on each task's behavior at a given time. After analyzing this study, we faced a problem with the authors.

They developed an observation subsystem that collects historical and hysteretic data by inspecting performance-monitoring counters and kernel data structures, gathering so information on a per-thread basis. They introduced OBS-X scheduling policy, that uses observations of each task's cache usage. OBS-X's goal is to distribute cache-heavy threads throughout the system, helping so to spread out cache load. When a new task is created, OBS-X looks for the LLC group with the smallest cache load, and places the new task in this group. OBS-X strength relates with the fact that this policy include the notion of overloaded tasks.

They ran two sets of experiments across four cores in two LLC groups. The first set of experiments consisted of four instances of cachebuster, an application that consumes as much cache as possible and four instances of spinloop, that consumes CPU with a minimum of memory access. They used [cb,sl][cb,sl] pairing, which represents the worst performance because both cachebuster applications contend for cache resources at the same time. With the addition of OBS-X, cachebuster performance increased between 12 % and 62 %, comparing with the default Linux default load balancing. The reason for the increase is that OBS-X distributed the cache-heavy tasks across LLC groups, thus minimizing the scheduling of heavy tasks together. To approximate real-world workloads, they ran OBS-X with a set of applications from the SPEC CPU 2000 suite run. The overall speedup increases to 4.6 %.

3.2 Vector Balancing Scheduling Policy

This policy reduces contention by migrating tasks, led by the information of *task activity vector* [18], that represents the utilization of chip resources caused by tasks. Based on the information provided form these vectors, it has been proposed from A. Merkel, J. Stoess and F. Bellosa [4] the scheduling policy that avoids contention for resources by co-scheduling tasks with different characteristics. The definition of activity vectors requires the read of a small number of the performance-monitoring counters (PMC) and asymmetric observations. This policy can be easily integrated in the OS balancing policy, so we can exploit the existing strategies. The weakness of this proposed solution by A. Merkel, J. Stoess and F. Bellosa [4] lies in the fact that these authors to avoid complexity in their research, assumed that tasks do little I/O, do not communicate with each other, they are independent. They used compute-intensive tasks. This assumption is a weakness because it limits the space where the Vector Balancing can be applied successfully. If there is communication, co-scheduling based on resource utilization can have conflicting goals. This is a topic of future work.

3.3 DIO (Distributed Intensity Online) Scheduler

S. Zhuravlev, S. Blagodurov and A. Fedorova [2] proposed DIO contention-aware scheduler. DIO scheduler continuously monitors the miss rates of applications, as we

argued in Sector 2 that it was the best contention predictor, then finds the best performance case and separates threads. It obtains the miss rates of applications dynamically online via performance counters. This makes DIO more attractive since the stack distance profiles, which require extra work to obtain online, are not required. Furthermore, the dynamic nature of the obtained miss rates makes DIO more flexible to application that have a change in the miss rate due to LLC contention. DIO was experimented in AMD Opteron with 8 cores, 4 cores for each domain. DIO improved performance by up to 13 % relative to default. Another use of DIO is to ensure QoS (Quality of Service) for critical applications, since it ensures to never select the worst performance case of the scheduler.

4 Adaptation of Contention-Aware Schedulers for NUMA Systems

Research studies on contention-aware algorithms, were primarily focused on UMA (Uniform Memory Access) systems, where there are multiple shared last level caches (LLC), but they have only one memory node associated with a memory controller, and the memory can be accessed with the same latency from every core. Modern multicore systems are using massively the NUMA (Non Uniform Memory Access) architecture, because of its decentralized and scalable nature. In these systems there is one memory node for each memory domain. Local nodes can be accessed for a shorter time than the distant ones, and each node has its own controller. According to S. Blagodurov, S. Zhuravlev, M. Dashti and A. Fedorova [1], when existing contention-aware schedulers designed for UMA architectures, were applied on a NUMA system (illustrated on Figure 3 [1]), they did not effectively manage contention, but they also degraded performance compared with the default contention-unaware scheduler (30% performance degradation).

4.1 Why existing Contention Management Algorithms degrade Performance on NUMA Systems?

S. Zhuravlev, S. Blagodurov and A. Fedorova [2] proposed DIO contention-aware scheduler. DIO scheduler continuously monitors the miss rates of applications, as we argued in Sector 2 that it was the best contention predictor, then finds the best performance case and separates threads. It obtains the miss rates of applications dynamically online via performance counters. This makes DIO more attractive since the stack distance profiles, which require extra work to obtain online, are not required. Furthermore, the dynamic nature of the obtained miss rates makes DIO more flexible to application that have a change in the miss rate due to LLC contention. DIO was experimented in AMD Opteron with 8 cores, 4 cores for each domain. DIO improved performance by up to 13 % relative to default. Another use of DIO is to ensure QoS

(Quality of Service) for critical applications, since it ensures to never select the worst performance case of the scheduler.

4.2 DINO Contention-Management Algorithm for NUMA Systems

As argued above, previous contention-aware algorithms were valid only on UMA architectures, but when applied to NUMA architectures, used in today's modern multicore processors hurt their performance. To address this problem, a contention-aware algorithm on a NUMA system must migrate the memory of the thread to the same domain where it migrates the thread itself. However, the need to move memory along with the thread makes thread migrations costly. So the algorithm must minimize thread migrations, performing them only when they are likely to significantly increase performance, and when migrating memory it must carefully decide which pages are most profitable to migrate. These are the challenges of designing a new contention-aware scheduling algorithm, which is appropriate with NUMA architecture. These challenges are handled in the study of S. Blagodurov, S. Zhuravlev, M. Dashti and A. Fedorova [1]. They have designed and implemented Distributed Intensity NUMA Online (DINO).

DINO scheduler uses the same heuristic model for contention as the DIO (Distributed Intensity Online) scheduler discussed in Section 3.3, that uses the *LLC miss rate* criteria for predicting contention. First of all, DINO tries to co-schedule threads of the same application on the same memory domain, provided that this does not conflict with DINO's contention-aware assignment. This is true for many applications [14]. DINO organizes threads in broad classes according to their miss rates as shown in the research study of Y. Xie and G. Loh [10]. The classes in which threads get divided are:

- Turtles: less than 2 LLC miss rates for 1000 instructions
- Devils: 2-100 LLC misses for 1000 instructions
- Super_Devils: more than 100 LLC misses for 1000 instructions

So the migrations will be performed only when threads change their classes, while they preserve their thread-core affinity relation as much as possible. For multithreaded applications DINO tries to co-schedule threads of the same application, in the same memory domain, but always avoiding to create conflicts in DINO's definitions regarding contention management. It also uses techniques to evaluate if it is convenient to co-schedule threads in the same domain or it would be better to separate them? DINO in this situation should at least avoid memory migration back and forth, preventing so performance degradation. DINO achieves this by separating threads in classes as explained above.

Results of DINO implementation showed that DINO achieved up to 30 % performance improvements for jobs in the MPI workload.

4.3 AMPS the Scheduling Algorithm for Performance-Asymmetric Multicore System NUMA & SMP

Since industry is going towards multicore technology, and traditional operating systems are based on homogenous hardware, and performance-asymmetric architectures (or heterogeneous) [21][23], present a very convenient solution regarding the cost they have, it appears the necessity to setup the relation between two different technologies. As a first step towards this, T. Li, D. Baumberger, D. A. Koufaty and S. Hahn [6] designed the operating system scheduler AMPS, that manages efficiently both SMP and NUMA-style performance-asymmetric-architectures. AMPS contains three components:
- Asymmetry-aware-load-balancing, that balances threads to cores in proportion with their computing power
- Faster-core-first scheduling, that controls thread migrations based on predictions of their overhead.

Our evaluation demonstrated that AMPS improved stock Linux for asymmetric systems in the aspect of performance and fairness.

AMPS uses thread-independent policies, which schedule threads independently regardless of application types and dependencies. This is considered a weakness that should be eliminated in the future. Thread-dependent policies mostly exists in research. H. Zheng, J. Nieh [24] dynamically detect process dependencies to guide scheduling.

5 Related Works

Research on solutions for the problem of resource contention on multicore systems is wide and dates back many years. Initial research in this field, were based on the idea that the primary factor on degradation performance in such systems was contention for shared cache. We mention the study of J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, P. Sadayappan [5] who evaluated the impact of existing cache partitioning methods on multicore system performance. They observed with most workloads, a significant performance improvement (up to 47 %). The only limitation of this study was that their experiments were limited by the hardware platform they used.

S. Zhuravlev, S. Blagodurov, A. Fedorova [2] through extensive experimentation on real systems, determined that along with it, other factors like memory controller contention, memory bus contention and prefetching hardware contention all combine in complex ways to create the performance degradation. They proposed DIO which improved performance by up to 13 % relative to the default operating system scheduler. Prior to DIO, were proposed also other scheduling policies like OBS-X [3], which uses the operating systems observations of behavior of threads running in the systems and then makes a decision on how to migrate threads for a better performance; All these studies were primarily focused on UMA systems, while

DINO contention-aware scheduler, remains the most appropriate until today for NUMA and uses miss rate as a contention predictor, like DIO does.

Prior research demonstrated that compared to homogeneous ones, asymmetric architectures deliver higher performance at lower costs in terms of die area and power consumption. T. Li, D. Baumberger, D. A. Koufaty, S. Hahn [6] proposed AMPS scheduler that manages efficiently both SMP and NUMA-style performance-asymmetric architectures. The problem of contention of heterogeneous architectures is almost uncovered, that is why it is a field of future research.

6 Conclusions and Discussions

Based on the wide dissemination of multicore processors, we chose to handle the topic of contention for shared resources in such systems, as it affects directly their performance. One of the major difficulty encountered during design of such schedulers, was selecting the most effective thread classification scheme, used to choose the best performance case respective to a specific pairing of co-scheduled threads. To mitigate contention for shared resources, we discussed and reviewed the best scheduling algorithms and policies, that do not perform equally when applied to different multicore architectures. So for UMA systems, we reviewed OBS-X scheduling policy, that uses the operating system dynamic observations on tasks behavior to migrate threads; Vector Balancing scheduling that takes migration decisions based on the task activity vector information and DIO contention-aware scheduling which is the best solution for UMA, because it mitigates contention for all shared resources, not only for cache contention, as OBS-X does. Moreover, Vector Balancing provides a limited solution, as it is based on compute-intensive and independent tasks, that do little I/O. These previously proposed contention-aware scheduling policies applied to NUMA modern multicore systems proved to hurt these systems' performance, because they fail to eliminate memory controller contention and create additional interconnect contention, that is why they needed adaptation to this new architecture. The most appropriate solution for NUMA systems is the DINO contention-aware scheduler, as it solves the performance degradation problem associated with the previous contention-aware solutions by migrating the thread along with its memory and also eliminates superfluous migrations. AMPS is the first scheduler proposed for the performance-asymmetric architectures, that supports both NUMA and SMP-style performance-asymmetric architectures, but it does not completely address contention, requiring further research in the future.

References

1. Blagodurov, S., Zhuravlev, S., Dashti, M., Fedorova, A.: A Case for NUMA-aware Contention Management on Multicore Systems. In: The 2011 USENIX Annual Technical Conference, pp. 1-9 (2011).

2. Zhuravlev, S., Blagodurov, S., Fedorova, A.: Addressing Contention on Multicore Processors via Scheduling. In: Proceedings of ASPLOS, pp.1-6 (2010).
3. Knauerhase, R., Brett, P., Hohlt, B., Hahn, S.: Using OS Observations to Improve Performance in Multicore Systems. In: IEEE Micro 28, 3 , pp. 54-58 (2008).
4. Merkel, A., Stoess, J., Bellosa, F. : Resource-Conscious Scheduling for Energy Efficiency on Multicore Processors. In: Proceedings of EuroSys, pp.6-8, pp.11-13 (2010).
5. Lin, J., Lu, Q., Ding, X., Zhang, Z., Zhang, X., Sadayappan, P.: Gaining Insights into Multicore Cache Partitioning: Bridging the Gap between Simulation and Real Systems. In: Proceedings of International Symposium on High Performance Computer Architecture, pp. 1-5 (2008).
6. Li, T., Baumberger, D., Koufaty, D.A., Hahn, S.: Efficient Operating System Scheduling for Performance- Asymmetric Multi-core Architectures. In: Proceedings of Supercomputing, pp.1-4, pp.8-10 (2007).
7. Zhang, X., Dwarkadas, S., Shen, K.: Towards practical page coloring-based multicore cache management. In: Proceedings of the 4th ACM European Conference on Computer Systems 2009.
8. Qureshi, M. K., Patt, Y. N.: Utility-based cache partitioning: A low overhead, high-performance, runtime mechanism to partition shared caches. In MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture , pp. 1-3 (2006).
9. Chandra, D., Guo, F., Kim, S., Solihin, Y. : Predicting InterThread Cache Contention on a Chip Multi-Processor Architecture. In HPCA '05: Proceedings of the 11th International Symposium on High Performance Computer Architecture (2005).
10. Xie, Y., Loh, G.: Dynamic Classification of Program Memory Behaviors in CMPs. In: Proceeding of CMP-MSI, pp. 2-4 (2008).
11. Blagodurov, S., Zhuravlev, S., Fedorova, A.: Contention-aware Scheduling on Multicore Systems. ACM Trans. Comput. Syst. 28 (December 2010).
14. Zhang, E. Z., Jiang, Y., Shen, X.: Does Cache Sharing on Modern CMP Matter to the Performance of Contemporary Multithreaded Programs? In: Proceedings of PPOPP (2010).
16. Jiang, Y., Shen, X., Chen, J., Tripathi, R.: Analysis and Approximation of Optimal Co-Scheduling on Chip Multiprocessors. In: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT '08), pp. 220-229 (2008).
17. Fedorova, A., Seltzer, M.I, Smith, M.D.: Improving Performance Isolation on Chip Multiprocessors via an Operating System Scheduler. In: Proceedings of the Sixteenth International Conference on Parallel Architectures and Compilation Techniques (PACT'07), pp.25-38 (2007).
19. Zhang, X., Dwarkadas, S., Folkmanis, G., Shen, K.: Processor Hardware Counter Statistics as a First-Class System Resource. In: Proceedings of the 11th USENIX workshop on Hot topics in operating systems (HOTOS'07).
20. McGregor, R. L., Antonopoulos, C. D., Nikolopoulos D. S.: Scheduling Algorithms for Effective Thread Pairing on Irbid Mutiprocessors. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05).
21. Shelepov, D., Saez Alcaide, J.C., Jefferym S., Fedorova, A., Perez, N., Huang, Z. F., Blagodurov, S., Kumar, V.: A Scheduler for Heterogeneous Multicore Systems. In: SIGOPS Operating Review,43(2) (2009).
22. Antonopoulos, C., Nikolopoulos, D., Papatheodorou, T.: Scheduling Algorithms with Bus Bandwidth Considerations for SMPs. In: International Conference on Parallel Processing (October 2003).

23. Balakrishnan, S., Rajwar, R., Upton, M., Lai, K.: The Impact of Performance Asymmetry in Emerging Multicore Architectures. In: Proceedings of the 32th Annual International Symposium on Computer Architecture, pp. 506-517 (June 2005).
24. Zheng, H., Nieh, J.: A Scheduler with Automatic Process Dependency Detection. In: Proceedings of the First Symposium on Networked Systems Design and Implementation, pp. 183-196 (March 2004).
25. Suh, G. E., Devadas, S., Rudolph, L.: A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning. In: Proceedings HPCA'02, pp.117-128 (2002).

S. Markovski, M. Gusev (Editors): ICT Innovations 2012, Web Proceedings, ISSN 1857-7288