# Improved Resource Allocation Algorithms for Practical Image Encoding in a Ubiquitous Computing Environment

Mianxiong Dong, Long Zheng, Kaoru Ota, Song Guo
School of Computer Science and Engineering, The University of Aizu
Aizu-Wakamatsu 985-8580, Japan
Email: mx.dong@ieee.org, {m5112105, d8102104, sguo}@u-aizu.ac.jp

Minyi Guo, Li Li
Department of Computer Science and Engineering, Shanghai Jiao Tong University
Shanghai, 200030, China
Email: {guo-my, lilijp}@cs.sjtu.edu.cn

***Abstract*—As a case study of the ubiquitous computing system, we have implemented a prototype for the JPEG encoding application. In order to achieve this eventual development in the real world, we studied resource allocation policies that can improve the overall performance of the system. In this paper, we consider those static and dynamic allocation approaches and then propose four different allocation algorithms. In particular, we extensively studied the dynamic allocation algorithms by exploring various cache policies which include disabled cache, unrestricted cache and restricted cache. Performance of these algorithms in large scale application scenario is also evaluated based on both the improved prototype and a simulation environment. The experimental results show a significant performance improvement achieved by the new proposed algorithms in terms of load balance, execution time, waiting time and execution efficiency.**

***Index Terms*—Resource allocation algorithm, caching, pervasive computing**

## I. INTRODUCTION

The word *ubiquitous* means an interface, an environment and a technology that can provide all benefits in a transparent manner anytime and anywhere [1]. Ubiquitous computing is a concept that computing facilities are available everywhere in the real world [2]. In recent years, ubiquitous devices such as RFIDs, sensors, cameras, T-engines, and wearable computers have been consistently upgraded and have begun to play important roles in our daily life [3-5]. However, there are still many technical challenges to build such applications that potentially exist in nearly every aspect of lives over infrastructure-less networks.

Olympus Future Creation Laboratory and University of Aizu have conducted a collaborative research on developing a general framework for the coming ubiquitous society, in which a ubiquitous computing scenario named Ubiquitous Multi-Processor (UMP), which is supported by many heterogeneous processing nodes, has been extensively studied. In order to evaluate the scalability and performance of the heterogeneous multiprocessor systems, a basic framework of multiprocessor simulation system has been implemented based on a multi-way cluster [6] and a double-buffered communication model [7] has been incorporated into the system that can improve the performance over 50%, in terms of communication speed, independent of various types of individual processors. We have extended the system and implemented a ubiquitous multi-processor network-based pipeline processing framework [8], at the hardware simulation level, to support the development of high performance pervasive applications. As a special case, the distributed JPEG encoding application has been successfully developed upon the proposed framework. The performance of this practical image encoding application has been evaluated in [9-10] and the optimal packet size of the UMP network been found through experiments in the UMP system.

In order to further improve the performance of this application for its practical deployment, we shall extend our previous work [9-11] by exploring various resource allocation techniques. In this paper, we propose a group of resource allocation algorithms and evaluate their performance in terms of load balance of the Resource Router (RR), total execution time, execution efficiency and task waiting time (delay).

The remainder of this paper is structured as follows. Section 2 gives the architecture of UMP system. Section 3 discusses the existing resource allocation algorithm and proposes three improved algorithms. The implementation details and performance evaluation are shown in Section 4. Section 5 summarizes our findings and the directions for the future work.

## II. AN OVERVIEW OF THE UBIQUITOUS MULTI-PROCESSOR SYSTEM

The architecture of our UMP system is illustrated in Fig. 1, in which there are three types of nodes: Client Node, Resource Router and Calculation Nodes. As the

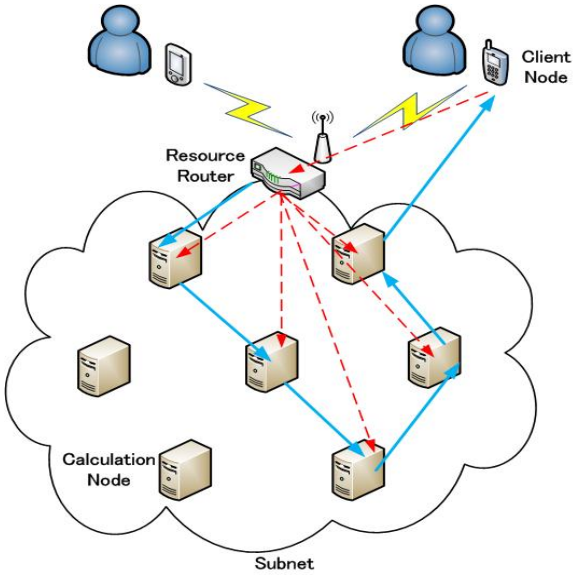



Figure1. The architecture of the UMP system based on our current implementation

mobile terminals, Client Nodes send task requests to the UMP system through a wireless network. The Resource Router is the gateway of the UMP system which receives requests from the Client Nodes and manages the corresponding tasks to be executed over the subnet. There exists only one Resource Router in a subnet. Each task can be decomposed into steps, each of which is executed on a specific Calculation Node in the subnet, such that the whole task can be accomplished by a set of Calculation Nodes that are cooperated and organized in a sequential manner. Various services/tasks thus can be supported by different execution sequences of the Calculation Nodes.

As a case study of the UMP system, we implemented a prototype for the application of the JPEG encoding [8], which is to convert bitmap format image to JPEG format image with six steps. They are reading bitmap file, RGB to YCbCr, down sampling translator, processing Discrete Cosine Transform, Huffman Encoding, and JPEG image writer. At the beginning stage of the implementation, the scheduling algorithm is not the major optimization issue because it has little performance effect in a small-scale task request scenario. As we extend this prototype to the real-world JPEG encoding application, in which many users would request the tasks to the UMP system simultaneously, some improved scheduling algorithms should be carefully designed to achieve good performance, e.g. load balancing, high execution efficiency and short waiting and execution time.

## III. Resource Allocation Algorithms

### A. A Preliminary Algorithm

In this paper, we use JPEG encoding as an archetypal example to test the proposed algorithms. There are six stages to encode a bitmap file into JPEG image. When a user requests a task of JPEG encoding, the RR will first reserve six PEs as a chain for the whole processing. After the user connected to the first PE, the chain processing will be started. When the last PE finished its sub-task, the user can get the result and the RR change the entire PEs chain to a standby status. Due to the user side is assumed as a mobile client, the battery life-time is a very important factor in the system design. To reduce the energy consumption of user side, we fix the first PE and the last PE to provide the frequently access from user to search the last PE. Thus, all the optimization process is effect to the middle PEs in the whole process chain. Therefore, the algorithm can be described as follows.

**Static Allocating Algorithm (SA).** When task comes, RR will reserve the whole PEs which will be needed to process the task until the task is finished. During the processing time, even some PEs are free, they cannot be used by other tasks.

The characteristic of the current resource allocation algorithm can be analyzed into two parts:

i) Mean delay:

$$d = \frac{1}{n}\left( m \cdot \sum_{i=1}^{\lfloor n/m \rfloor} (i-1)t + (n - \lfloor n/m \rfloor m) \cdot \lfloor n/m \rfloor t \right) \quad (1)$$

where $m$ is the number of tasks RR can handle at one time, $t$ is the time to handle $m$ tasks.

So the first $m$ tasks wait 0 time, the second $m$ tasks wait $t$ time, the $i$-th $m$ tasks should wait $(i\text{-}1)t$ time.

We can also get task execution efficiency as follows:

ii) Task Execution Efficiency:

$$f = \sum_{i=1}^{n} e_i \Big/ \left( \sum_{i=1}^{n} e_i + \sum_{i=1}^{n-1} c_{i,i+1} \right) \quad (2)$$

Where $e_i$ $(1 \le i \le n)$ is the execution time in $i$-th PE, $c_{j,j+1}$ is the communication time between $j$-th PE and $(j+1)$-th $(1 \le j \le n)$ PE.

In our simulation, we assume the communication time between any two PEs is the same, i.e. $c_{i,j} = c_{m,n} = c \forall i, j, m, n \in N$ , $N$ is the natural number set. Hence,

$$f = \sum_{i=1}^{n} e_i \Big/ \left( \sum_{i=1}^{n} e_i + (n-1) \cdot c \right) \quad (3)$$

Abstract of the SA is described as follows.

(1) Router retrieves a new task from the task queue. If there is no task in the task queue, then ends.
(2) Router generates a PE chain which is used to process a task. Set the PEs in the PE chain as busy, which means these PE can not do any other task until they are released.
(3) Router sends the PE chain information and task to PE1.
(4) PE1 finishes its work and follows the PE chain information to transfer the task to PE2.
(5) PE2 finishes its work and follows the PE chain information to transfer the task to PE3.
(6) PE4 finishes its work and follows the PE chain information to transfer the task to PE5.
(7) PE5 finishes its work and follows the PE chain information to transfer the task to PE6.
(8) PE6 finishes its work and sends the processed task back to router.
(9) Router sets all PEs in the PE chain as idle.
(10) Remove the task from the task queue. If there is no task left in the task queue, then terminates. Otherwise go to Step (1).

*B. Improved Algorithms*

**Dynamic Allocating Algorithm (DA).** The biggest limitation of the current policy is that if the RR allocates the PEs to the users once, the all PEs are reserved until the whole task will be finished. This is obviously a big useless of the computational resource. To regard as this point, we apply a randomly distribute algorithm to the UMP system. The concept of DA is after the PE finished the execution of the process, the PE will ask the RR for the next phase of PE. The usage rate of PE is quite high, but the load balance is heavy for the RR. We can get task execution efficiency as follows:

i) Task execution efficiency:

$$f = \sum_{i=1}^{n} e_i \Big/ \left( \sum_{i=1}^{n} e_i + \sum_{i=1}^{n-1} cr_i + \sum_{i=2}^{n} c_{i-1,i} \right) \quad (4)$$

where $e_i 1 \le i \le n$, is the execution time, $c_{i-1,i} 2 \le i \le n$ is the communication time between *i*-th PE and RR, $cr_i 1 \le i \le n$ is the communication time between *(i-1)*-th PE and *i*-th PE.

Abstract of the DA is described as follows.

(1) Router retrieves a new task from the task queue. If there is no task in the task queue, then ends.
(2) Router finds an idle PE1 and any PE6 and then transfers the task to this PE1.
(3) After getting the task from router, this PE1 sends a busy status message to router.
(4) After processing the task, this PE1 send an idle status message to router and meantime ask the router for the next PE.
(5) Router finds an idle PE2, and then tells the PE1.
(6) PE2 sends the status busy to router; PE1 transfers the task to PE2, and sends idle status message to router.
(7) PE2, PE3, PE4 act the same.
(8) After PE5's processing the task, PE5 transfers the task the PE6 which is decided by router at Step 2.
(9) After PE6's processing the task, transfer the processed task back to router.
(10) Remove the task from the task queue. If there is no task left in the task queue, then terminates. Otherwise go to Step (1).

**Dynamic Allocating Algorithm with Cache Technology (DA-C).** To improve the DA, we introduce a cache concept of the resource allocating algorithm. For every PE, we assign a cache for them to memorize the next stage's PE. When they finished their sub-task, they will search the next phase of PE in their cache. If the all PEs in the cache are at the busy status, it will ask RR to assign one free PE as the next phase PE.

We can get task execution efficiency as follows:

i) The best case of task execution efficiency:

$$f = \sum_{i=1}^{n} e_i \Big/ \left( \sum_{i=1}^{n} e_i + \sum_{i=2}^{n} (3c_{i-1,i}) \right) \quad (5)$$

where $e_i 1 \le i \le n$ is the execution time, $c_{i-1,i} 2 \le i \le n$ is the communication time between *(i-1)*-th PE and *i*-th PE. The best case means each *(i-1)*-th PE can access each *i*-th PE memorized in their cache because *i*-th PE is not busy. *(i-1)*-th PE is supposed to have communication with *i*-th PE three times in total. As the first communication, *(i-1)*-th PE asks *i*-th PE whether or not it is busy currently. Then, *i*-th PE replays to *(i-1)*-th PE in the second communication. Since this is the best case so that *i*-th PE's answer must be "available", *(i-1)*-th PE starts to send data to *i*-th PE in the third communication.

ii) The worst case of task execution efficiency:

$$f = \sum_{i=1}^{n} e_i \Big/ \left( \sum_{i=1}^{n} e_i + \sum_{i=2}^{n} (3c_{i-1,i}) + \sum_{i=1}^{n-1} cr_i \right) \quad (6)$$

where $e_i 1 \le i \le n$ is the execution time, $cr_i 1 \le i \le n-1$ is the communication time between *i*-th PE and RR, $c_{i-1,i} 2 \le i \le n$ is the communication time between *(i-1)*-th PE and *i*-th PE. The worst case means each *(i-1)*-th PE cannot access each *i*-th PE memorized in their cache because *i*-th PE is busy. Therefore, each *(i-1)*-th PE has to ask RR for a free PE.

Abstract of the DA-C is described as follows.

(1) Router retrieves a new task from the task queue. If there is no task in the task queue, then ends.
(2) Router finds an idle PE1 and any PE6 and then transfers the task to this PE1.
(3) After getting the task from router, this PE1 sends a

busy status message to router.
(4) After processing the task, this PE1 sends an idle status message to router.
(5) Find the idle PE from its cache.
(6) If the PE1 finds an idle PE2, then send a request message to verify whether the PE2 is truly idle or not.
(7) If the response from PE2 is yes, then go to step 10; or send a request message to router, by which router will look for an idle PE2 without restriction of PE1's cache.
(8) If router finds one, then tell PE1, otherwise, repeat steps from step 5.
(9) Router finds an idle PE2, and then tells the PE1.
(10) PE2 send the busy status message to router; PE1 transfers the task to PE2, and then sends the idle status message to router.
(11) PE2, PE3, PE4 act the same, besides updates the information of cache of PE1, PE2, PE3, respectively.
(12) After PE5's processing the task, PE5 updates the information of cache of PE4; and then transfers the task the PE6 which is decided by router at Step 2.
(13) After PE6's processing the task, transfer the processed task back to router.
(14) Remove the task from the task queue. If there is no task left in the task queue, then terminates. Otherwise go to Step (1).

**Dynamic Allocating Algorithm with Restricted Cache (DA-RC).** We introduce a restrict cache concept to DA-C. The difference between the DA-RC and DA-C is the restriction of jumping to the PEs which is out of the cache that current PE has. That means when a certain PE finished its sub-task and the whole PEs in the cache are busy, the PE will not ask RR to assign a free PE out of the cache. For example, assume every cache at each phase has four PEs. When a certain PE at one stage finished the sub-task, it can search the next phase PE in the same cache. If the next phase PE is all busy status, it has to wait. This is the biggest difference between DA-C and DA-RC. And the Efficiency of DA-RC in the best case is the same to DA-C. Here, the best case means each *(i-1)*-th PE succeeds to find the next phase PE in only one access without asking all PE members in the cache.

i) The worst case of task execution efficiency:

$$f = \sum_{i=1}^{n} e_i \Bigg/ \left( \sum_{i=1}^{n} e_i + \sum_{i=2}^{n} \left( \sum_{j=1}^{l} 2c_{i-1,j} + c_{i-1,i} \right) \right) \qquad (7)$$

where $e_i 1 \le i \le n$ is the execution time, $c_{i-1,i} 2 \le i \le n$ is the communication time between *(i-1)*-th PE and *i*-th PE, $c_{i-1,ij} 2 \le i \le n, 1 \le j \le l$ where l is the number of PEs in one group, is also the communication time between *(i-1)*-th PE and *i*-th PEs in the group. The worst case means each *(i-1)*-th PE asks all next phase PEs in a group at every stage. It is because *(i-1)*-th PE checks whether or not PE in the group is busy one by one in order to seek one available PE.

Abstract of the DA-RC is described as follows.

(1) Router retrieves a new task from the task queue. If there is no task in the task queue, then ends.
(2) Router finds an idle PE1 and any PE6 and then transfers the task to this PE1.
(3) After getting the task from router, this PE1 sends a busy status message to router.
(4) After processing the task, this PE1 sends an idle status message to router.
(5) Find the idle PE from its cache.
(6) If the PE1 finds an idle PE2, then send a request message to verify whether the PE2 is truly idle or not.
(7) If the response from PE2 is yes, then go to step 10; or waits a particular time, then go to step 5.
(8) PE1 transfers the task to PE2.
(9) PE2, PE3, PE4 act the same, besides updates the information of cache of PE1, PE2, PE3, respectively.
(10) After PE5's processing the task, PE5 updates the information of cache of PE4; and then transfers the task the PE6 which is decided by router at Step 2.
(11) After PE6's processing the task, transfer the processed task back to router.
(12) Remove the task from the task queue. If there is no task left in the task queue, then terminates. Otherwise go to Step (1).

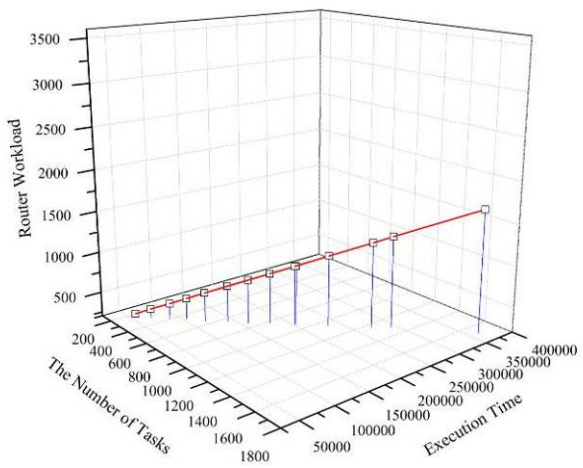
## IV. Performance Evaluation and Discussion

### A. Detail of the Implementation

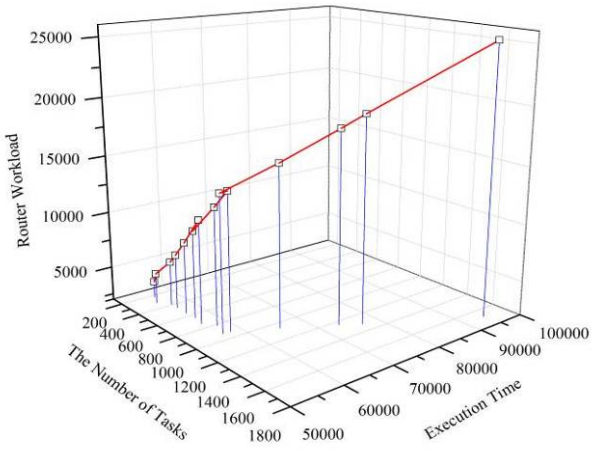
We built a simulation system to evaluate the four algorithms. Theoretically, the number of tasks arriving at the UMP system during each period is a random number with an upper bound from 8 to 60 and a lower bound 0. We simulates 50 periods. Therefore, on average, the total number of tasks is about from 200 to 1500. Nevertheless, in our implementation, we ran the simulation in which the number of tasks is from 200 to 1800 with about every 100 interval. The number of PE was set as 144. Because the JEPG encoding needs 6 steps to process, each task needs six PEs; therefore the total chains of PEs are 24. We also set the network delay in which a PE or Router sends a request or gets a response as 20, and the network delay in which a PE or Router send or receive the raw JEPG as 200. Table 1 shows the environment of the simulation.

TABLE I
THE EXPERIMENT ENVIRONMENT

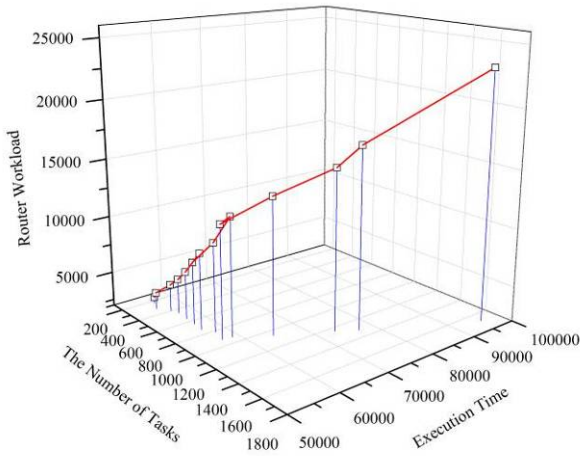
| | |
|---|---|
| OS | Windows Vista Business (32-bit) |
| CPU | AMD Athlon64 3200+ |
| Memory | DDR SDRAM 2GB |
| Language | JAVA 1.5 |
| Network | Localhost |



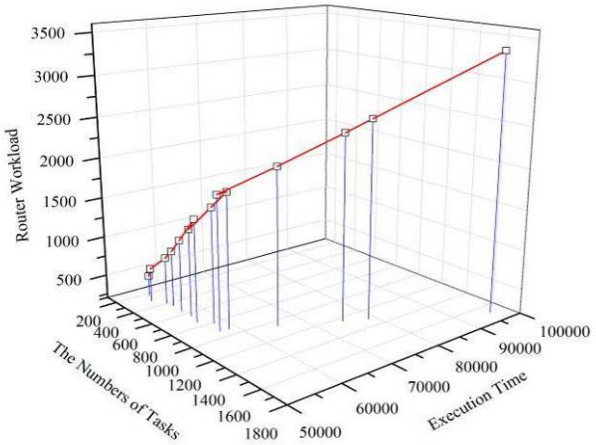

The SA algorithm




The DA algorithm




The DA-C algorithm




The DA-RC algorithm

Figure 2. The workload of RR of various algorithms

### *B. Simulation Results and Discussion*

We use three dimensional figures to provide an overall aspect in terms of number of tasks, execution time and resource workload. Here, we define the router's workload as that every time each PE sends a request to the RR, we count the workload as 1. Fig. 2 shows the workload of RR from the simulation results. Left axis indicates the numbers of tasks and right axis indicates the execution time. Vertical axis shows RR's workload. The workload of SA is obviously small than DA and DA-C because once the RR assign the PE to execute the task, it will never communicate with PEs. But when we focus on the total execution time, SA performs the worst result. We can find its execution time is almost four times comparing to the other three algorithms. Even the shape of the red lines of DA, DA-C and DA-RC in these pictures are alike, we can easy to know that by using the cache technology, DA-RC performs an extremely good result than DA and DA-C and nearly close to the SA in router workload. DA-RC's router workload is only 12.5% of DA and DA-C's.

Fig. 3 which is the two dimensions of view of router workload also shows the significance improvement of DA-RC. It is nature that the DA had bad result, because almost every time the PE should ask RR to know the next phase PE which should be connected to.

In Fig. 4, task execution efficiency is highly related with the waiting time, SA shows the worst result with the reason that it has to wait the execution to start even there are free PEs in the process chain. We can see the execution efficiency of DA-RC has 12.6% better than DA. Also, the curves of DA-C and DA-RC are almost the same and they are exactly matching the mathematic model we have described in the above section.

Delay (waiting time) is an important factor in the real world system. Supposed even the total executions time of the system is good, but if the delay is huge, the system still cannot be well used by users. It is very clear that the average of delay of SA is extremely large because the

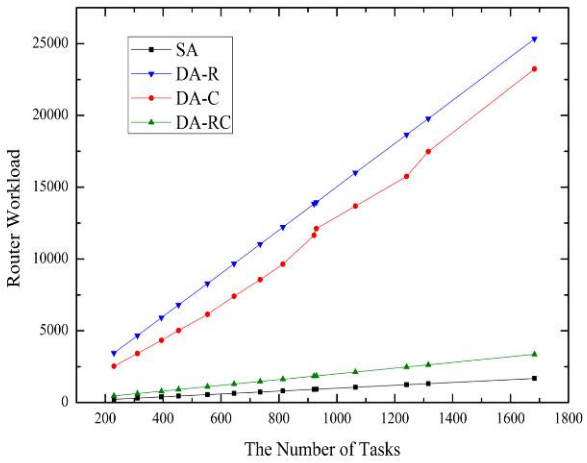



Figure 3. Algorithm VS (Router Workload)

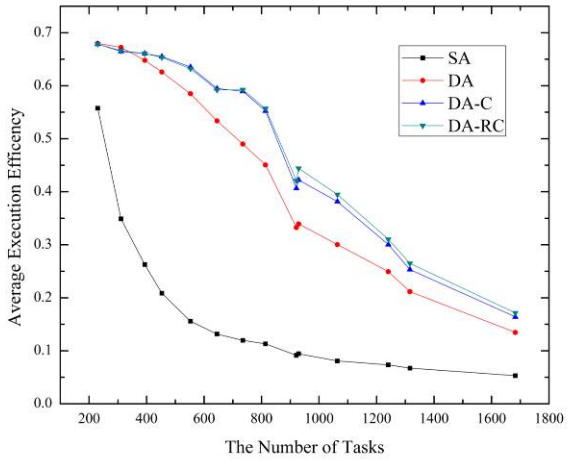



Figure 4. Algorithm VS (Average Execution Efficiency)

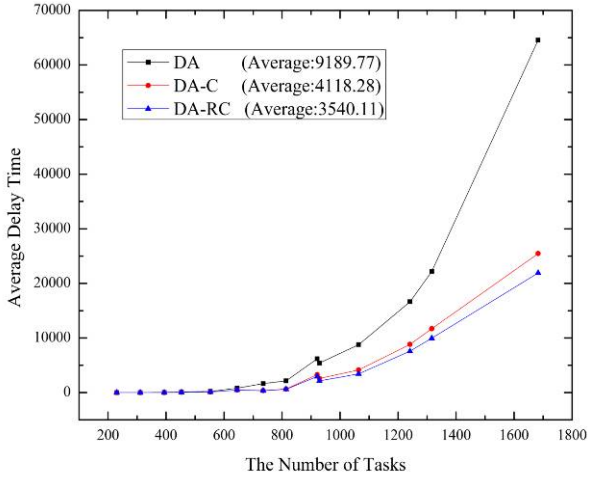



Figure 5. Algorithm VS (Average Delay)

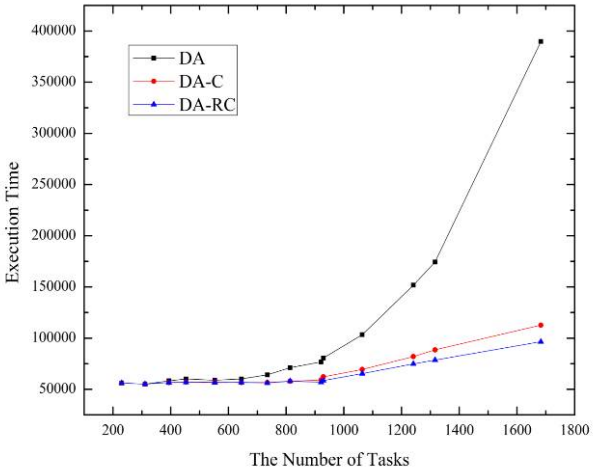



Figure 6. Algorithm VS (Execution Time)

execution procedure is almost the sequential. Hence, we omit it in Fig. 5 to prevent its negative influence on other curves of the three algorithms. The average delay of SA is 60194, DA is 9189, DA-C is 4118 and DA-RC is 3540. From the figure, we can find DA-RC and DA-C improved much better performance than DA in terms of average delay time. The reason why DA-RC is slightly better than DA-C can be considered that in DA-RC the waste of the fail communication time is omitted.

DA-C and DA-RC show good performance again in Fig. 6. From the figure, we can know DA boost the execution time in an exponential manner. It is hard to accept this algorithm for practical usage. On the other hand, DA-C and DA-RC remain slow increasing even the number of task becomes larger. Comparing to the algorithms with each other, DA-RC is the 15% better than DA-C at the number is 1700. It is obvious that DA-RC will overwhelmed the DA-C when the scale of the system goes bigger and bigger.

Fig. 7 to 10 are the results comparing the cases when size of cache is 1, 2 and 4. We can know the execution efficiency and delay time is always good when the cache size is 4. From Fig. 7, the Router Workload is the same and at the early stage of the number of tasks of Fig. 8, 9 and 10, we can see the performance is almost the same. However, as the number of tasks grows, the 4-size-cache case is showing a better result than the other two cases.

## V. Conclusion and Future Work

As the further step of our previous works, we studied resource allocation policies that can improve the overall performance of UMP system. In this paper, we considered those static and dynamic allocation approaches and then proposed four different allocation algorithms. We evaluated the performance with four different resources allocating algorithms and analyzed these algorithms from four points of view in terms of load balance, execution time, waiting time and execution efficiency. Through these extensive experiments, we have successfully validated our proposed algorithms achieved significant performance improvement. We found the Dynamic Allocating Algorithm with Restricted Cache is the best algorithm to allocate resources (PEs) under condition that the system has many users and many tasks to deal with. And the next better algorithm is DA-C,

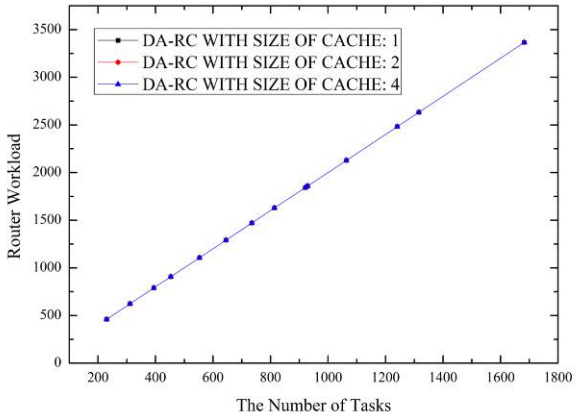



Figure 7. Router Workload under various cache sizes

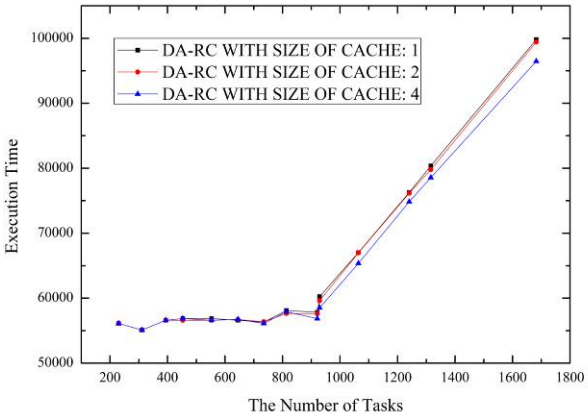



Figure 8. Execution Time under various cache sizes

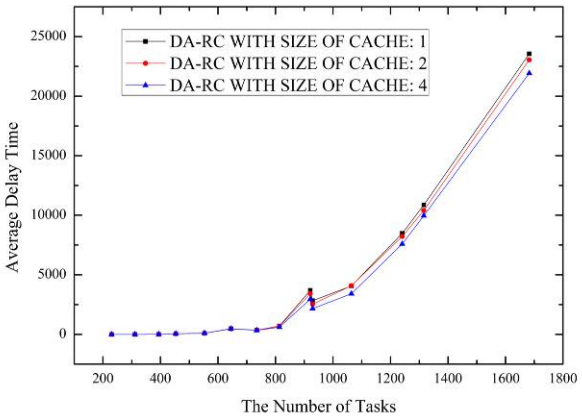



Figure 9. Average Delay Time under various cache sizes

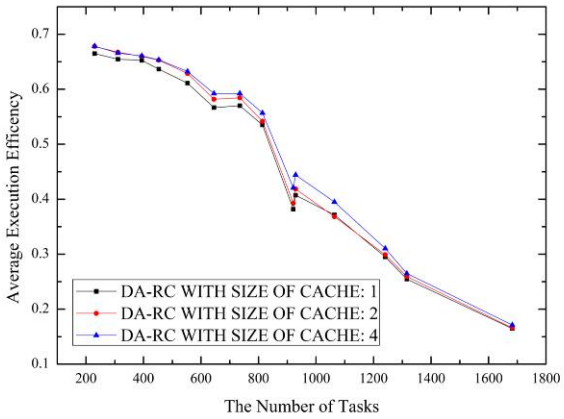



Figure 10. Average Execution Efficiency under various cache sizes

followed by DA. Furthermore, through the experiments we have realized that we can set the allocating policy flexibly to answer the users' requests. In the future, we shall consider the multi-layer architecture of the resource router to reduce the workload and also focus on the upper layer of the UMP system - the service layer, to deal with the context information from users, resources and environments.

## ACKNOWLEDGEMENT

This work is supported by Future Creation Lab., Olympus Corp, Research Fellowships of the Japan Society for the Promotion of Science for Young Scientists Program, the National High-Tech Research and Development Plan of China (863 Plan) under Grant Nos. 2008AA01Z106, the National Natural Science Foundation of China under Grant Nos. 60811130528, 60725208, and 60533040, and Shanghai Pujiang Plan No. 07pj14049.

The authors are deeply grateful to Mr. Deze Zeng, Mr. Gongwei Zhang and Mr. Peng Li in the Performance Evaluation Laboratory at the University of Aizu.

## REFERENCES

[1] M. Weiser, "The Computer for the 21st Century", IEEE Pervasive Computing, pp. 19-25, January-March 2002.

[2] Wikipedia, http://ja.wikipedia.org/wiki/

[3] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges", IEEE Personal Communication, pp. 10-17, August 2001.

[4] S. R. Ponnekanti, et.al, "Icrafter: A service framework for ubiquitous computing environments", *in Proc. of Ubicomp* 2001, pp. 56–75, Atlanta, Georgia, October 2001.

[5] V. Stanford, "Using Pervasive Computing to Deliver Elder Care", IEEE Pervasive Computing, pp. 10-13, January-March, 2002.

[6] A. Shinozaki, M. Shima, M. Guo, and M. Kubo, "A High Performance Simulator System for a Multiprocessor System Based on a Multi-way Cluster", Advances in Computer Systems Architecture, Lecture Notes in Computer Science vol. 4186, pp. 231-243, Springer Berlin/Heidelberg, September, 2006.

[7] A. Shinozaki, M. Shima, M. Guo, and M. Kubo, "Multiprocessor Simulator System Based on Multi-way Cluster Using Double-buffered Model", *in Proc. of IEEE AINA* 2007, pp. 893-900, Niagara Falls, Canada, May 2007.

[8] M. Kubo, B. Ye, A. Shinozaki, T. Nakatomi and M. Guo, "UMP-PerComp: A Ubiquitous Multiprocessor Network-Based Pipeline Processing Framework for Pervasive Computing Environments", *in Proc. of IEEE AINA* 2007, pp. 611-618, Niagara Falls, Canada, May 2007.

[9] M. Dong, S. Guo, M. Guo and S. Watanabe, "Design of the Ubiquitous Multi-Processor System Focusing on Transmission Data Size", *in Proc. of HPSRN*, pp. 158-166,

Sendai, Japan, March 2008.
[10] M. Dong, S. Watanabe, and M. Guo, "Performance Evaluation to Optimize the UMP System Focusing on Network Transmission Speed", *in Proc. of FCST*, pp. 7-12, Wuhan, China, November 2007.
[11] M. Dong, M. Guo, L. Zheng, S. Guo, "Performance Analysis of Resource Allocation Algorithms Using Cache Technology for Pervasive Computing System", *in Proc. of ICYCS* 2008, pp. 671-676, Zhang Jia Jie, China, November 2008.

**Mianxiong Dong** received the B.S. and M.S. degree both in computer science and engineering from the University of Aizu, Japan, in 2006 and 2008 respectively. He is currently a Ph.D. student and a JSPS (Japan Society for the Promotion of Science) Research Fellow at School of Computer Science and Engineering, the University of Aizu, Japan. From January 2007 to March 2007, he was a visiting scholar of West Virginia University, USA. His research interests include pervasive computing, sensor networks, and ubiquitous-learning.

**Long Zheng** received the B.S in computer science and technology from Huazhong University of Science and Technology, China, in 2006. He is now a Master student at School of Computer Science and Engineering, the University of Aizu, Japan. His research interests include chip multiprocessor (CMP), pervasive computing and Peer-to-Peer media streaming.

**Kaoru Ota r**eceived the B.S. degree in computer science and engineering from the University of Aizu, Japan, in 2006 and M.S. degree in computer science at Oklahoma State University, USA, in 2008. She is currently a Ph.D. student at School of Computer Science and Engineering, the University of Aizu, Japan. Her current interests of research are localization and tracking by using mobile agents in wireless sensor networks.

**Song Guo** received the PhD degree in computer science from the University of Ottawa, Canada in 2005. He then held a position with the University of British Columbia on an NSERC (Natural Sciences and Engineering Research Council of Canada) postdoctoral fellowship. From 2006 to 2007, he was an Assistant Professor at the University of Northern British Columbia, Canada. He is currently an Assistant Professor at School of Computer Science and Engineering, the University of Aizu, Japan. His research interests are in the areas of protocol design and performance analysis for communication networks, with a special emphasis on wireless ad hoc and sensor networks for reliable, energy-efficient, and cost effective communications.

**Minyi Guo** received the PhD degree in computer science from University of Tsukuba, Japan. Before 2000, Dr. Guo had been a research scientist of NEC Corp., Japan, and a professor in the School of Computer Science and Engineering, The University of Aizu, Japan. Currently, Dr. Guo is a distinguished chair professor of the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China and an adjunct professor at the University of Aizu. He is also a guest professor at Nanjing University, Huazhong University of Science and Technology, and Central South University, China. Dr. Guo has published more than 160 research papers in international journals and conferences. Dr. Guo has served as general chair, program committee, or organizing committee chair for many international conferences. He is the founder of the International Conference on Parallel and Distributed Processing and Applications (ISPA) and the International Conference on Embedded and Ubiquitous Computing (EUC). He is the editor-in-chief of the Journal of Embedded Systems. He is also on the editorial board of the Journal of Pervasive Computing and Communications, the International Journal of High Performance Computing and Networking, the Journal of Embedded Computing, the Journal of Parallel and Distributed Scientific and Engineering Computing, and the International Journal of Computer and Applications. Professor Guo received the National Science Fund of China (NSFC) for Distinguished Young Scholars in 2007, and is also the PI of the NSFC Key Project "Theoretical and Technical key points of Pervasive Computing." Dr. Guo's research interests include parallel and distributed processing, parallelizing compilers, pervasive computing, embedded systems software optimization, and software engineering. He is a senior member of the IEEE and the IEEE Computer Society, and a member of the ACM, IPSJ, CCF, and IEICE.

**Li Li** received the M.E. degree in computer science and engineering from the University of Aizu, Japan in 2005. Her employment experience included the department of information physics of Nanjing University, China, the national institute for environmental studies Tsukuba, Japan, respectively. Li Li has worked for the school of software of Shanghai Jiao Tong University as engineer since 2006. Her research interests include pervasive computing, sensor networks, and ubiquitous-learning.