Pipeline Based Resource Allocation Design in Ubiquitous Communication Networks

Mianxiong Dong, Kaoru Ota, Long Zheng, Gongwei Zhang and Minyi Guo

Abstract— **Pervasive computing has become a very hot research field. In our previous work, we have proposed a system named UMP-Percomp, a Ubiquitous MultiProcessor-based Pipeline processing architecture to support high performance pervasive application development. So far we have implemented a prototype system to evaluate the performance of the architecture. However, the structure of the prototype system has some limitations: lack of scalability, inefficient resource allocation algorithm, and lack of flexibility for new kind of tasks. Hence we solve these problems and improve the current UMP system. We add a UDP server to each component to support scalability and component substitution. We also design a parallel algorithm to maximum the usage of PEs (Processing Elements), and we also consider the situation of lacking PEs suitable for requested tasks. Finally, we run extensive experiments on the new system to compare with the old system.**

Index Terms—**pipeline-based; scalability; resource allocation; pervasive computing**

I. Introduction

As hardware industry develops rapidly, people now can share many kinds of computers; can share many servers; can have a desktop computer and a notebook, and our house can have many embedded devices. We now have come into one person, many computers era, that is ubiquitous computing era, or pervasive computing era [1]. Yet we still need to do many researches on it.

In order to create this future technology, Olympus Company and The University of Aizu cooperate with each other, and create a system called UMP (Ubiquitous Multi-Processor) System. We assume that, in the future more and more microprocessors with extremely limited resources will be embedded into pervasive computing environment. And under most circumstances, these microprocessors only have unit function. So in the UMP System, each of these processors is called PE (Processing Element). Different kind of PE has different function, several kinds of PE can do together to finish a task. There are so many kinds of PEs around you, in your office, in your home, or in your lab. So the objective of the UMP System is to organize these heterogeneous PEs and running pervasive computing application across these PEs. Besides, the UMP System also wants to support mobile computing. In order to achieve these purposes, we first designed a ubiquitous pervasive computing framework (full name is Ubiquitous Multi-Processor Network-Based Pipeline Processing Framework), and then implemented a UMP based JPEG encoding application [2]. The interesting place of JPEG encoding application is that: user takes a photo by a camera equipped mobile phone, and then asks the mobile phone to encode the photo into JPEG, and finally view the processed photo on the mobile phone. It seems that all the work is done on the mobile phone, but in fact, the mobile phone sends the encoding work to UMP System, and wait for the result returned by the UMP System. Due to this technology, we can do more complex task on mobile phone, and the duration of mobile phone's battery can be longer.

Though this seems very promising and interesting, there are some insufficient points on the designing and implementing of UMP System which we will formulate clearly later. In brief, there are mainly three problems of current system: one is lack of scalability and component substitution strategy; one is lack of efficient pipeline scheduling, and the last one is mainly about the ability to handle new kind of task. The main purpose of this research is to address the above mentioned questions to make the UMP System better. For the first question, we create a general model for each component in the UMP System, so we can easily implement scalability and component substitution. For the second question, we introduce a pipeline algorithm. As for the third question, we consider the handling of new kinds of task to make UMP System more ubiquitous, and make a user to have a better experience. Finally, we implement the modified system and evaluate it by huge experiments of simulation.

The remainder of this paper is organized as follows. Section 2 reviews related works. Section 3 shows overview of the current UMP system. Section 4 formulates three problems of the current system and their corresponding solutions are given in section 5. In section 6, we show experiment results and analysis followed by conclusion in section 7.

Manuscript received November 19th, 2010. This work is supported in part by Japan Society for the Promotion of Science (JSPS) Research Fellowships for Young Scientists Program, JSPS Excellent Young Researcher Overseas Visit Program and National Natural Science Foundation of China (NSFC) Distinguished Young Scholars Program (No. 60725208).

M. Dong, K. Ota and L. Zheng are with School of Computer Science and Engineering, the University of Aizu, Aizu-Wakamatsu, Fukushima, 965-8580, Japan (e-mail: mx.dong@ieee.org, k.ota@ieee.org, d8112104@u-aizu.ac.jp).

G. Zhang and M. Guo are with Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. (e-mail: zhanggw@sjtu.edu.cn, guo-my@cs.sjtu.edu.cn)

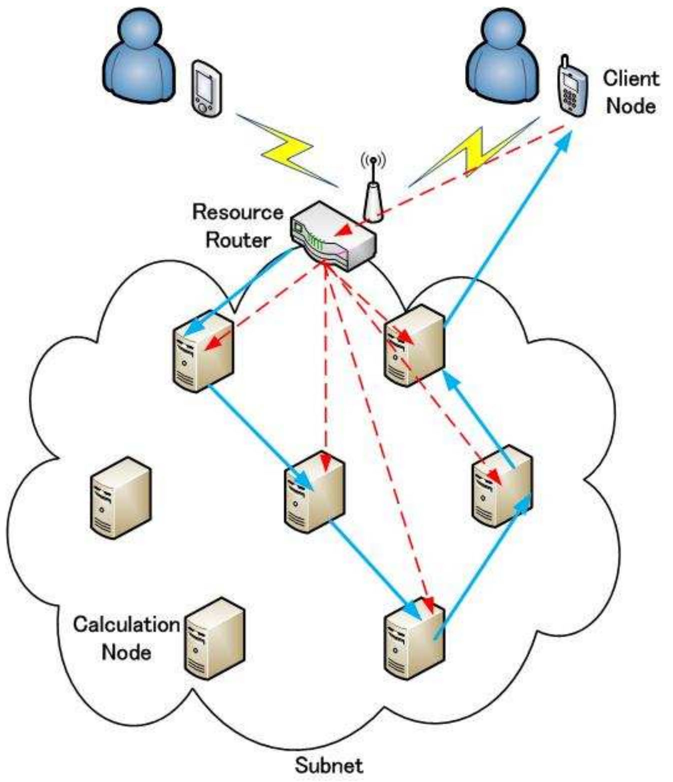

Fig. 1. Overview of the UMP system with one RR, two Client Nodes, seven Calculation Nodes.

II. Related Work

A lot of projects on pervasive computing have been done in the past few years. The solar project [4] in Dartmouth University is to build a graph-based abstraction for collecting, aggregating, disseminating context information. The project treats the context-aware application has "event-driven" structure, where event here refers to context information. And then they built a graph of operators. These events (context information) can flow through a directed acyclic graph of operators to subscribing applications. The one world project [5] provides an integrated, comprehensive framework for building pervasive applications. The project realizes the vision that the pervasive applications should continually adapt to an ever-changing environments and still function when people move through the physical world or switch devices. So they build the one world framework to facilitate developers to build pervasive applications. The Aura Project [6] aims at distraction-free. The project assumes that the bottleneck in computing is no longer CPU speed, disk capacity, or communication bandwidth. Instead the bottleneck is the limited resource of human attention. Researchers in this project build Aura framework to minimize user's attention and create an environments that adapts to the user's context and needs. The Oxygen Project [7] at MIT tries its best to be human-centered. The project treats computation has been centered about machines for the last forty years. People should interact with the machines on their term. The project assumes in the future, there will be abundant computation and communication, and these should be human-centered. They built Oxygen Project to make these computation and communication as pervasive and free as air, naturally into people's live. The Endeavour Project [8] tries to develop a revolutionary Information Utility, able to operate at planetary scale, in order to make it dramatically convenient for people to interact with information, devices and other people.

Researchers in Olympus Company and The University of Aizu also had done many works on pervasive computing previously. A. Shinozaki, M. Guo et.al developed a high performance simulator system-based on multi-way cluster [3]. In their simulator, they developed architecture on heterogeneous multiprocessor system, and use MPI library to implement inter-process communication in order to minimize CPU resource usage on communication wait state. At last, they develop a distributed JPEG Encoding application to test the performance. Inspired by the new insights in this work, M. Kubo, B. Ye et.al propose a UMP (Ubiquitous Multi-Processor) Framework to support pervasive computing environment and mobile computation [2]. M. Dong et.al test some parameters on the prototype system for optimizing [10][11].

III. An Overview of the Ubiquitous Multi-Processor System

A. Motivation of our work

The final goal of the UMP project is to provide a network framework for the coming ubiquitous era [13]. In the ubiquitous society, services are filled around the user just like the oxygen. Services are like the water come out from the faucet; they are everywhere and anytime to meet the user's requests. This trend requires computing resources with two opposing attributes: higher performance and lower power consumption. The UMP system has many processing elements (PEs). Each PE can have a particular function. To tell simply, UMP system is like a Tangram [12]. PE can be considered as a piece of the Trangram. Trangram could be many constructions. Using some pieces, we can get Rabbit or Yacht. As the same to it, The UMP system provides the various functions for the users just combine the precise PE and using it. In the ubiquitous society, users' needs are multifarious, so the best solution is to provide the several basic functions which can collaborate with others to provide various services. This idea brings the flexibility dynamics to the UMP system.

B. Big picture of the UMP system

In our system, there are three kinds of nodes as shown in Fig.1. One is the Client Node which works instead of the mobile users. This node requests tasks through mobile terminals on a wireless network. The other is the Resource Router (RR) which is a gateway of the system. There exists only one node of RR in one subnet. This node received task requests from the Client Nodes. Then, the node manages a list of tasks and determines which tasks should be executed currently on the subnet. The last one is the Calculation Node which actually executes tasks requested from the Client Nodes. Every task is allocated by the RR on the subnet. When a task is executed, several Calculation

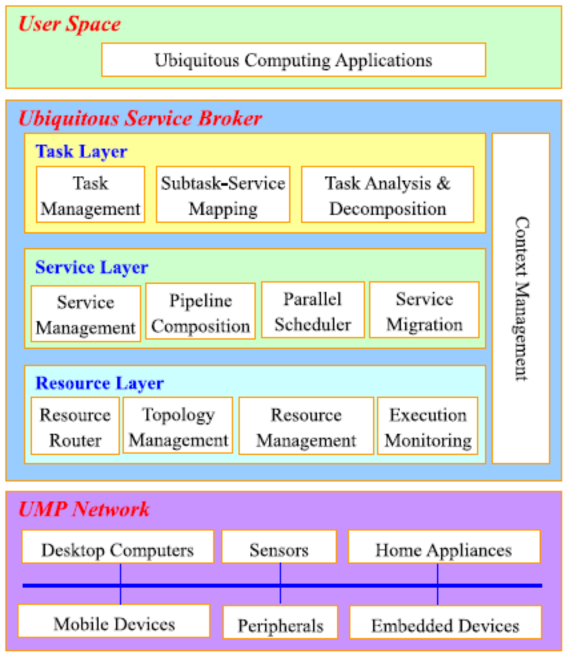

Fig. 2. Architecture of the UMP system

Nodes are connected to each other like a chain. For example, to encode a bitmap file into JPEG file, the step is 6. So it means the chain has 6 Calculation Node. Combination of the Calculation Nodes is always unique so that actions of tasks can be changed flexibly by demands of the Client Nodes.

C. *Architecture of the UMP system*

Fig.2. shows architecture of the UMP system proposed in our previous work [2]. In User Space, the ubiquitous application sends a task to Task Management in USB (Ubiquitous Service Broker). Then the Task Management asks Task Analysis & Decomposition component to decompose the task into small subtasks. Each subtask can only be done on corresponding PE. After decomposing, Task Layer sends the subtasks to Service Layer. Components in Service Layer will require PEs from Resource Layer, and schedule the PEs for subtasks. As for Resource Layer, it is responsible for managing PEs, including monitoring the state of PEs.

We assume every user has a mobile phone. And for the current UMP system, we only deployed it at one place and we focused on USB. Because USB is core part in the architecture, we will also focus USB on the improved system. It is a pity that we integrated all the components into RR in the current UMP system, but we will mend it in the improved system. Here is the description of how the current system works: The user uses the mobile phone to take a photo which is in raw and takes a large space. Then the ubiquitous application on this mobile phone connects to the RR on fixed UDP port. After connecting, the RR spawns a new client thread to communicate the client (here refers to the mobile phone) for current task. Then the mobile phone sends task (here refers to JPEG Encoding) and decomposed subtasks to the new client thread. The new client thread finds all the necessary PEs for these subtasks. If it cannot find them all, it rejects the task. If it finds them all, it sends the subtasks to the PEs, and any of these PEs cannot be allocated again for other tasks until all of these PEs freed by the system. Finally, the PE in the last step sends the result to the mobile phone.

IV. Problem Formulation

In this section, we define three problems to be solved in the current UMP system and corresponding solutions are given in the next section.

A. *Scalability*

Our current UMP system has a lack of scalability because of the following two sub-problems; component independent problem and component substitution problem

First, we describe what the component independent problem is. For simplicity, we only implemented some components of USB in the current UMP system. Moreover, these components are not independent. The RR component is responsible for the other components. So they must reside on the same server.

Second, to explain the component substitution problem, we take Task Manager Component as example. We have only a normal server, on which we deployed a Task Manager Component. This server can only handle a limited request from clients. Because in the ubiquitous society, it has a possibility that many users will use the system simultaneously, we want to add a more powerful server to the system, and deploy Task Manager Component on this server. However, this server has different IP address. So how can the client know the new server and send request to it? Another case is that the normal server comes down. Under such case you have to substitute it. However, the new server might have a new IP address. For example, we tested to deploy the Task Manager Component on an existing server which has a different IP address. As a result, the client fails to send the request.

B. *Resource Allocation*

The big limitation of the current policy is 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 can apply a pipeline based algorithm to the UMP system. Due to the user side is assumed as a mobile client, however, the battery life-time is a very important factor in the system design. Hense, we need to design an efficient resource allocation algorithm with consideration of reducing the energy consumption of user side.

C. *Handling diversity of tasks*

In the current system, we only deploy PEs for a specified kind of task, e.g. JPEG Encoding. So if a client requests a task but all the PEs are not subtasks of it, the PEs cannot deal with the task so that the system just reject it. However, practically, we will have many different kinds of users and each of them have many different kinds of requests (tasks). In addition, although all users request the same tasks, available PEs are not enough and some of the requests are rejected if the number of the users is

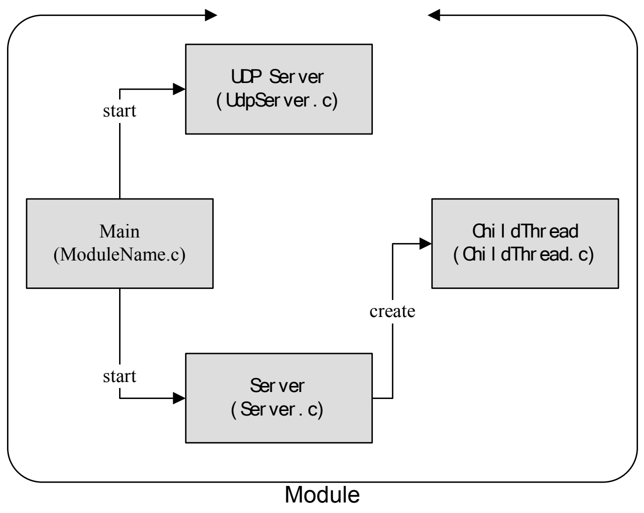

Fig. 3. Component Structure

quite large.

V. Design of the New UMP System

In this section, to design the new UMP system, we give solutions to the problems mentioned in the previous section. We firstly describe a new component structure to solve the first problem and propose resource allocation algorithms followed by introducing Emply PEs to the system, which can deal with diverse tasks.

A. New component structure

To solve the component independent problem, we implement the components of USB independently. That means each component can deployed on different servers. They do not have to be at the same server.

The key point to solve the second sub-problem is to ignore the concrete IP address of each Component. Inspired by the implementation of RR in the current system, we proposed a UDP based solution. We designed a UDP server for each component. The UDP server has a fixed port number, just as ftp has a well-know port number 25. Different components have different port numbers.

When there is a need for a component (or a client) to communicate with another component. It first creates a broadcast packet which contains a type and a random value. The type is used for identifying the component with which it wants to communicate. Because the receiving component may receive different broadcast packets from different components, it's better to use a value to differentiate these packets. After creating a proper packet, the sending component broadcasts the packet. On receiving this broadcasted packet, the receiving component check the type field in the packet, if it finds the packet is aiming for itself: it replies its own IP Address and other information like CPU speed to the sending component. Otherwise it just discards the packet. After broadcasting, the sending components can know the IP addresses and other information about the components to which it wants to connect. Base on the information like CPU speed, the sending component can choose a proper one to connect.

Now we solve the component independent problem. You may notify another problem has come up: how does a component know the port number of the other component? One method is to assume the port number is well-known. In order to consider the scalability of the system, we will use another strategy.

So far we only deployed the current system at one place, so we do not need to consider the migrate problem. If we want to run experiments on several places, we have to add a Task Migrate Component to the system. We only modify a little of the components that need to interact with Task Migrate Component. Task Migrate Component registers itself at a fixed map server. The map server allocates a unique port number for Task Migrate Component. So if a component wants to interact with Task Migrate Component. It just requires the corresponding port number for Task Migrate Component at map server for the first time. And then broadcasts to know the concrete IP address of a Task Migrate server. Next time, it needs not to require the map server again.

Considering the above problems, we propose a structure model for each component as shown in Fig.3. The UDP Server is used for replying IP address. Because there may be many connection requirements from other components, we also design a TCP server. It is used for handling requests, and creating a child thread for each request. In order to make the system more convenient, we integrate the starting of UDP Server of TCP Server into Main. So when you start a component, it can automatically start the corresponding TCP and UDP servers.

B. Resource Allocation Algorithms

We design two pipeline based algorithms, which are called as randomly allocating algorithm (RAA) and RAA with cache technology, such that the RR only allocates a necessary PE for current phase. Before describing two proposed algorithms, firstly we review the current algorithm as follows.

Current algorithm (CA): 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 can be analyzed into two parts:

Mean delay:

$$d = \frac{m \cdot 0t + m \cdot 1t + m \cdot 2t + \cdots + m \cdot (\left\lfloor \frac{n}{m} \right\rfloor - 1)t + (n - \left\lfloor \frac{n}{m} \right\rfloor m) \cdot \left\lfloor \frac{n}{m} \right\rfloor t}{n} \qquad (1)$$

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-1)t* time. According to these, we can compute the mean delay(*d*) time as follows:

$$d = \frac{\sum_{i=0}^{\left\lfloor \frac{n}{m} \right\rfloor} it + (n - \left\lfloor \frac{n}{m} \right\rfloor m) \cdot \left\lfloor \frac{n}{m} \right\rfloor t}{n} \qquad (2)$$

We can get task execution efficiency as follows:

Task Execution Efficiency:

$$f = \frac{\sum_{i=1}^{n} e_i}{\sum_{i=1}^{n} e_i + \sum_{i=1}^{n-1} c_{i,i+1}} \tag{3}$$

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

In our simulation, we assume the communication time between any two PE 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. So,

Task execution efficiency:

$$f = \frac{\sum_{i=1}^{6} e_i}{\sum_{i=1}^{6} e_i + (n-1) * c} \tag{4}$$

Randomly allocating algorithm (RAA): We apply a randomly distribute algorithm to the UMP system. 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. The concept of RAA 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:

Task execution efficiency:

$$f = \frac{\sum_{i=1}^{n} e_i}{\sum_{i=1}^{n} e_i + cr_1 + \sum_{i=2}^{n} (2 * cr_i + c_{i-1,i})} \tag{5}$$

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

Randomly allocating algorithm with cache technology (RAA-C): To improve the RAA, we introduce the cache technology. For every PE, we assign a cache for them to memorize the next stage's PE. When they finish their sub-task, the 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:

Task execution efficiency:

$$f = \frac{\sum_{i=1}^{n} e_i}{\sum_{i=1}^{n} e_i + \sum_{i=2}^{n} (3 c_{i-1,i})} \tag{6}$$

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 *ith* PE.

C. Available PE and Empty PE

To solve the third problem, we have the following two kinds of PEs: Available PE and Empty PE. Available PE carries an execution code to do a specific subtask and cannot work for different kind of subtask. In order to complete a task requested by a user, we need a group of Available PEs for all the subtasks composing the task. We call them a set of Available PEs. On the other hand, Empty PE does not carry any code, so it cannot do any kind of subtask. It however can load any kind of codes from a repository PE or a repository Server, and then can do the corresponding subtask.

We utilize Empty PEs when we cannot find any Available PE in the case when Available PEs are all busy or they cannot deal with a requested task. In other word, unless we have no Empty PEs, otherwise we will never give up to fulfill a task from a user request.

The strategy for handling diversity of tasks is as follow:

(1) Find a set of Available PEs (It does not mean that they are allocated at a time, just for testing the requirement). If find all, return OK; else if cannot find all, but have empty PEs, go to step (2); otherwise reject the task.
(2) Ask the system for necessary codes, if find all, loading the codes to Empty PEs, otherwise go to step (3)
(3) Ask the client for necessary codes (it is handled by the ubiquitous application, it is opaque to any user), if find all, loading the codes to Empty PEs, otherwise go to step (4)
(4) Check if the task is a migrating task, if yes, borrow the code from last place; if no, and reject the task.

The users are unconscious of all the above steps; therefore the users experiment better performance without rejected by the system.

VI. PERFORMANCE EVALUATION AND ANALYSIS

In this section, we evaluate performance of the UMP system in terms of impact on resource allocation algorithms as well as impact with Empty PEs. Simulation results of them are shown in the following subsections respectively.

A. Performance impact on resource allocation algorithms

For the first experiment, we built a simulation system to evaluate the three algorithms. We used Poisson Distribution [14] for task generation to bring the simulation close to the real environment. Because The Poisson Distribution arises in connection with Poisson processes. It applies to various phenomena of discrete nature whenever the probability of the phenomenon happening is constant in time or space. The number of tasks is from 2500 to 5500. We run the simulation of the number of tasks from 2500 to 5500 with every 150 interval. We set number of PE as 2400. Because the JEPG encoding need 6 steps to process, each task needs 6 PEs, therefore the total chains of PEs are 400. We also set the network delay as 100.

Fig. 4 shows the load balance of RR from the simulation result. The load balance of CA is obviously small than RAA and RAAC because once the RR assign the PE to execute the task, it will never communicate with PEs. So, we omitted the comparison in Fig. 4. From the picture, we can see by using the

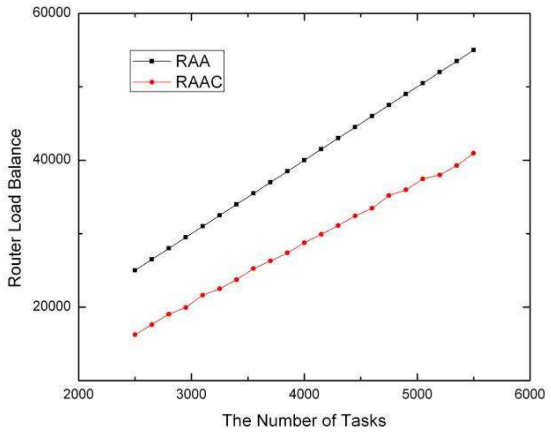

Fig. 4. The load balance of RR with RAA and RAAC

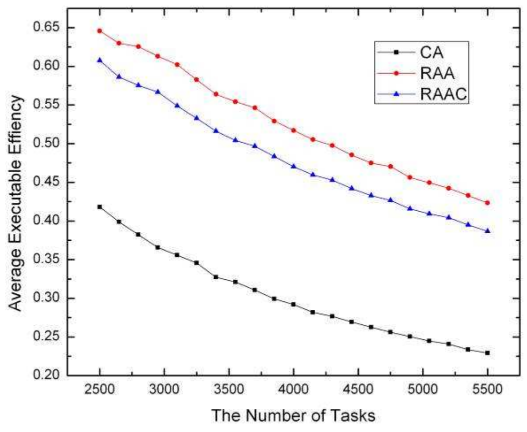

Fig. 5. The execution efficiency of tasks with CA, RAA, and RAAC

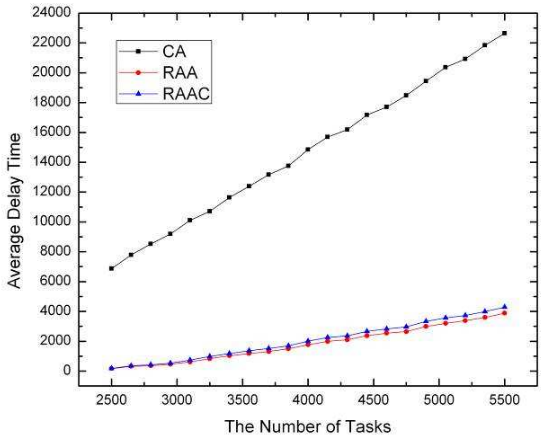

Fig. 6. The delay (waiting time) of tasks with CA, RAA, and RAAC

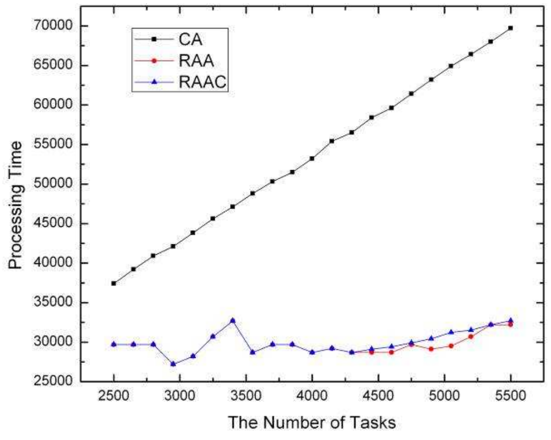

Fig.7. The total execution time of the tasks with CA, RAA, and RAAC

cache technology the summation of load balance of RR, RAAC perform a good result than RAA. And it is naturally the RAA 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.5, task execution efficiency is highly related with the waiting time, CA shows the worst result as well as it has to wait the execution to start even there are free PEs in the process chain. We can see RAA has a better result than RAAC. But consider the loading balance of RR it is still not acceptable.

Delay (waiting time) is an important factor in the real system. Supposed even the total executions time is good, but if the delay is very large, the system is still cannot be well used by users. From Fig. 6, we can see the average of delay of CA is extremely huge. That is because the execution procedure is almost the sequential. The reason of why RAA is slightly better than RAAC is RAA can fully randomly use the next phase of PE. So the waist of the fail communication time is omitted.

From the left part of Fig. 7, we can see the proposed algorithms have a significant improvement compared with the current implementation. From the bottom part of Fig. 5, we can see the RAA is slightly performing a good result than RAAC. But the difference is very small, so we can neglect in the real system implementation.

Though these experiments, we have successfully proofed our proposed algorithms are meaning while. Considering those four key factors (task execution efficiency, load balance of the RR, reducing the delay of task execution, total processing time of the whole task), both of RAA and RAAC has its meit and demerit. There is always a tradeoff between these factors. Taking into account the balanced point of all factors, we found the RAAC is much more suitable for the real environment to allocating resources (PEs) when the condition is the system has many users and many task to process. If the users and tasks are not so large RAA could have a good performance too. One more things we have realized through the experiments are we can set the allocating policy flexibly answering to the user’s request. Maybe that is the best solution to design the UMP system.

B. Performance impact with Empty PE

In the second experiment we use the same simulation system as the first experiment. We run the experiment under three different situations to evaluate .

For the first situation, we only offer Available PEs for one kind of task in each system. We also offer a lot of empty PEs and

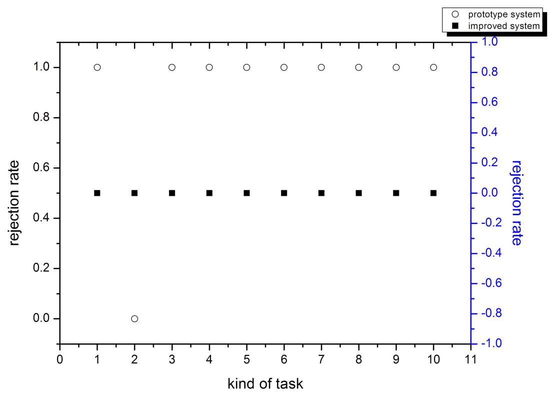

Fig. 8. Rejection rate with diversity of tasks

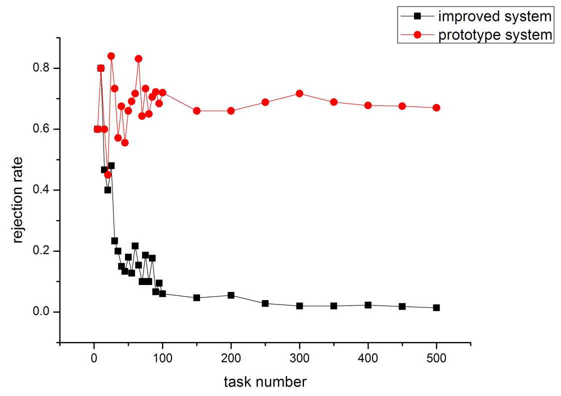

Fig. 9. Rejection rate over the number of tasks

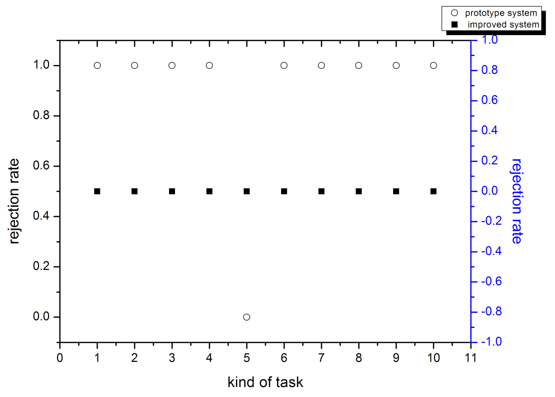

Fig. 10. Rejection rate with diversity of tasks when Empty PEs can retrieve any codes from anyplace

binaries in each system. These binaries can be registered by the administrator in the system. Then we sent ten kinds of tasks requests to each system. Fig. 8 shows a rejection rate of a requested task. We can see that in the current system (denoted by a circle), it only accepts task 2, but reject all other kinds of tasks. The reason is that we only include Available PEs for task 2. In the improved system (denoted by a square), it accepts all the tasks. Though we also only provide Available PEs for task 2, the Empty PEs can load necessary codes from some place to execute the new kind of task.

For the second situation, we also only offer Available PEs for one kind of task in each system, which means the system can only do a task at the beginning. We do not offer codes in each system this time. But we assume that the client has random kinds of codes, so the system can ask the client for codes. We only send three kinds of task requests to each system. But the same kind of task may be sent several times to the system. The number of the same kind task is randomly selected.

At first, there are only a few tasks to process. The kind of task is random selected. We know for both systems, they can only do one kind of task at the beginning. So if the number of this kind is large, the rejection rate will be low. Otherwise, the rejection rate will be high. In a word, the rejection rate seems stochastic. But as the number of tasks become more and more, the number of each kind will tend to be the same. So for the current system, after the number of tasks reaches 150, the rejection rates are going to be similar. That means the current system will reject two kinds of tasks and accepts only one kind. Because the number of each kind of task is almost the same, the reject rate is approximate to 2/3≈0.6667. As for the improved system, we know it can get new codes from client. As it handles with more clients, it gets more codes. As a result, it can do more kinds of tasks. So you can see in Fig. 9 that the rejection rate becomes lower as the number of tasks gets larger (We assume a client only do a few tasks in a limited time, this sounds reasonable). A perfect result is that the system accumulates all the necessary codes, and then it will accept any kind of task.

For the third situation, the unchangeable condition is that both systems can only handle a kind of task at first. And this time we consider the task migration. The improved system can borrow the codes from another place, while the current system cannot. We also do the ten kinds of tasks on both systems. The result is similar to the first experiment as shown in Fig. 10. In the current system, it only accepts tasks of kind 5. This is the only different from the first experiment, because in this time's experiment, the systems are provided Available PEs for kind 5 task. You know, the current system has no way to get proper PEs for other kinds of tasks. In the improved system, when a migrating task comes, it can always borrow the codes from last place. So we can accept all the migrating tasks in the new system.

VII. Conclusion

In this paper, we addressed three problems in the UMP System: lack of component substitution and scalability, lack of efficient resource allocation algorithms, high rejection rate for tasks requested by users. We design a UDP and TCP servers based structure to support component substitution and scalability. We proposed two different resource allocation algorithms to utilize PEs efficiently. Also, we involve two kinds of PEs to handle diversity of tasks. We ran a number of simulation experiments and results show execution efficiency can increase much more than the previous system and also rejection rate can decrease. As future work, we will extend the system to larger scale, and we will add more functions to it.

References

[1] Satyanarayanan, "*Pervasive Computing: Vision and Challenges*", *IEEE PCM August 2001, pp. 10 – 17*

[2] M. Kubo, B. Ye, A. Shinozaki, T. Nakatomi, M. Guo, "UMP-Percomp: A Ubiquitous Multiprocessor Network-Based Pipeline Processing Framework for Pervasive Computing Environments", *in Proc. of IEEE AINA'07*

[3] A. Shinozaki, M. Shima, M. Guo, and J. Kubo. "A high performance simulator system for a multiprocessor system based on multi-way cluster", *in Proc. of 2006 Asia-Pacific Conf. Computer System Architecture systems, pages 231-243, Shanghai, China, September 2006*

[4] G Chen, D Kotz, "Solar: A Pervasive-Computing Infrastructure for Context-Aware Mobile Applications", *Mobile Computing Systems and Applications, 2002. In Proc. of Fourth IEEE Workshop, Page 105-114*

[5] R.Grimm, "One world: Experiences with a pervasive computing architecture", *IEEE Pervasive Computing, 3(3):22-30, July 2004*

[6] D. Garlan, D. Siewiorek, A. Smailagic, and P. SteenKiste, "Project aura: Towards distraction-free pervasive computing", *IEEE Pervasive Computing, 21(2):22-31, April 2002.*

[7] Oxygen, MIT, http://www.oxygen.lcs.mit.edu

[8] Endeavour, UC Berkeley, http://endeavour.cs.berkeley.edu

[9] Helal, S. Winkler, B. Choonhwa Lee Kaddoura, Y. Ran, L. Giraldo, C. Kuchibhotla, S. Mann, W. "Enabling location-aware pervasive computing applications for the elderly", *in Proc. of IEEE Percom 2003 Page531-536, March 2003*

[10] M. Dong, S. Guo, M. Guo and S. Watanabe, "Design of the Ubiquitous Multi-Processor System Focusing on Transmission Data Size", *in Proc. of The 2008 International Workshop on High Performance and Highly Survivable Routers and Networks, pp. 158-166, Sendai, Japan, March 2008*

[11] M. Dong, S. Watanabe, and M. Guo, "Performance Evaluation to Optimize the UMP System Focusing on Network Transmission Speed", *in Proc. of Japan-China Joint Workshop on Frontier of Computer Science and Technology, pp. 7-12, Wuhan, China, November 2007*

[12] Tangram, Wikipedia, http://en.wikipedia.org/wiki/Tangram

[13] OLYMPUS Future Creation Laboratory : Fields of Research, Olympus Corporation, http://www.fc-lab.jp/en/activ/human/ubiquitous.html

[14] L. Kleinrock, Queueing Systems Volume 2: Computer Applications, John Wiley & Sons, 1979.