International Journal of Computer Theory and Engineering, Vol. 1, No. 5, December, 2009
1793-8201

A Linda-based Hierarchical Master-Worker Model

Mohammad GhasemiGol, Mostafa Sabzekar, Hossein Deldari and Amir-Hassan Bahmani

***Abstract*—In this paper we will implement a new version of master-worker architecture that improves the previous ones. The common Master-Worker paradigm consists of two entities: a master and multiple workers. The master is responsible for decomposing the problem into small tasks and managing them until all tasks are completed. Therefore, the master should endures heavy load either communication or computation. This bottleneck in the master process typically occurs when the number of workers increases because the master process will not be able to keep all workers equally busy. The paper presents a novel technique for hierarchically nesting the basic master-worker scheme. This technique resolves the said problem by presenting a hierarchical scheme and reduces the communicational messages due to the usage of the Linda model. The obtained results for large matrix multiplication case study on a real cluster show the effectiveness of our model.**

***Index Terms*—Hierarchical Master-worker, Linda model, Linda-based Submaster, Communication overhead.**

I. Introduction

Grid computing [1] has become an alternative to traditional supercomputing environments for developing parallel applications, in recent years. But, its building is more complex than traditional parallel computing environments. There are several high-level programming frameworks have been proposed to simplify the development of large parallel applications for Computational Grids (for example Netsolve [8], Nimrod/G [9], MW [10]).

The Master-Worker paradigm is a common model to evaluate a pool of tasks that is used by many scientific and engineering applications like tree search algorithms, genetic algorithms, training of neural networks, stochastic optimization, parameter analysis for engineering design and Monte Carlo simulation [2].

In the simplest version of master-worker model we just have one master that produces tasks and many workers that do these tasks. Therefore, the master will be busy all the time while workers are idle. So the master is bottleneck. During the time some researchers struggle to improve this version. These efforts led to Hierarchical Master-Worker Skeletons [3] to decrease the load of master. In this model they investigate techniques for hierarchically nesting the basic master-worker scheme. It presents a skeleton implementation for nesting several master-worker instances. With this scheme the administrative load of task handling to a whole hierarchy of masters. The hierarchies have been elegantly expressed as foldings over the modified basic schemes.

But a problem is seen yet. If the number of workers or submasters grows, the submasters also will be bottleneck because many communications appear between workers and their submasters. In this paper we introduce a new architecture for hierarchical master-worker to decrease the communication cost. For this purpose we define submasters as shared spaces which can be accessed by their own workers. We use the Linda space to implement these shared areas. In this architecture, several workers can refer to a submaster concurrently and many communications will be eliminated.
In general, the performance of master-worker applications will depend on the temporal characteristics of the tasks as well as on the dynamic allocation and scheduling of processors to the application.

In evaluating common master-worker architectures, two performance measures of particular interest are speedup and efficiency. Speedup is defined, for each number of processors n, as the ratio of the execution time when executing a program on a single processor to the execution time when n processors are used. Ideally, we would expect that the larger the number of workers assigned to the application the better the speedup achieved. The efficiency measure is the utilization of the n allocated processors. It is defined as the ratio of the time that n processors spent doing useful work to the time those processors would be able to do work. Efficiency will be a value in the interval [0,1]. If efficiency is becoming closer to 1 as processors are added, we have linear speedup. This is the ideal case, where all the allocated workers can be kept usefully busy. In this work we used these measures to evaluate our proposed architecture.

The rest of this paper is organized as follows. In Section 2 we discuss the common master-worker models in the literature. Section 3 reviews the Linda model at a glance. The structure of our Linda-based master-worker is presented in Section 4. We illustrate the effectiveness of our proposed model in Section 5. The conclusion is given in Section 6.

II. Review of the Common Master-Worker Models

The master-worker model is a simple scheme in which each processor is designated as number of workers, similar to the system suggested by Andrews and Polychronopoulos [4].

Manuscript received June 25, 2009.

Mohammad GhasemiGol is with the Department of Computer Engineering, Ferdowsi University of Mashhad (FUM), Mashhad, Iran (phone: +98-915-9620831; fax: +98-561-4434070).

Mostafa Sabzekar is with the Department of Computer Engineering, Ferdowsi University of Mashhad (FUM), Mashhad, Iran (phone: +98-160; fax: +98-561-4447178).

Hossein Deldari is with the Department of Computer Engineering, Ferdowsi University of Mashhad (FUM), Mashhad, Iran (phone: +98-915-3101510).

Amir-Hassan Bahmani was with the Department of Computer Engineering, Islamic Azad University of Mashhad (IAUM), Mashhad, Iran (phone: +98-915-1225532).

Workers simply perform given operations, while masters are responsible for preparing work for the workers and correlating their output into a global result. Masters are required to partition the problem-space, schedule work, and balance the load of the workers to maintain efficiency [5].

Space-based architecture (SBA) is a tool for implementation of a master-worker style application. The activities of each worker process in the system are coordination by a shared dependency graph. The dependency graph stores the current state of an application's execution, and is used by workers to determine which task should be executed next. The graph itself is a directed acyclic graph, with vertices representing an application's tasks, and the edges denoting the data dependencies between them. So, before a worker obtains a task for execution, it first takes the dependency graph from the space to see which task it should execute. The worker then takes this task from the space, and marks the corresponding node in the dependency graph as being in-progress. Also, a worker will use the graph to determine if the task depends on the results of any previously executed task, and, if so, will obtain these results before executing the current task. When a worker has completed the execution of a task, it will obtain the dependency graph and mark the node as complete, before returning the results of the task's execution to the space. Workers continue this process until all nodes in the graph are marked as complete, at which time the master process takes all of the results from the space and assembles them into some meaningful whole, depending on the particular application [6]. Fig.1 demonstrates a detailed master-worker model in this implementation.

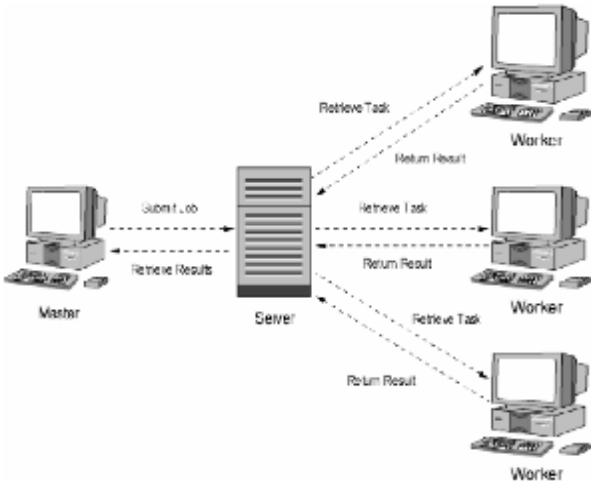


Fig. 1: Master-worker implimentation in SBA

MW is another tool for making a master-worker style application that works in the distributed, opportunistic environment of Condor. MW applications use Condor as a resource management tool, and can use either Condor-PVM or MW-File a file-based, remote I/O scheme for message passing. Writing a parallel application for use in the Condor system can be a lot of work. Since the workers are not dedicated machines, they can leave the computation at any time. In MW the master class manages a list of uncompleted tasks and a list of workers. The default scheduling mechanism in MW is to simply assign the task at the head of the task list to the first idle worker in the worker list.

However, MW gives flexibility to the user in the manner in which each of the lists is ordered. For example, MW allows the user to easily implement both a Last-In-First-Out policy (LIFO) and a First-In-First-Out policy (FIFO) by simply specifying the location at which new tasks are added to the task list to be one of add at end or add at begin in the method [7]. Details of this architecture are shown in Fig. 2.

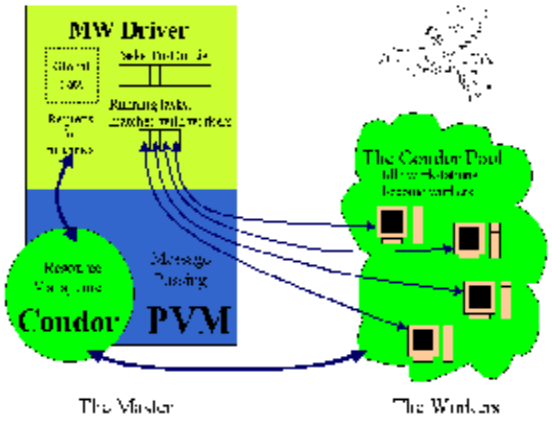


Fig. 2: Relationships between Condor, PVM, and the MWDriver

Berthold et al. proposed hierarchical master-worker skeletons. In this model they investigate techniques for hierarchically nesting the basic master-worker scheme. It presents a skeleton implementation for nesting several master-worker instances. With this scheme the administrative load of task handling to a whole hierarchy of masters. The hierarchies have been elegantly expressed as foldings over the modified basic schemes. A simple structure of this paradigm is shown in Fig. 3.

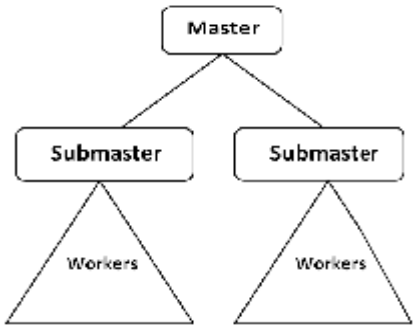


Fig. 3: Hierarchical master-worker system

The main problem of master-worker models is that the master is busy all the time. Hierarchical master-worker model use submasters to decrease the workload of the master. Fig. 4 shows a typical activity profile for a master-worker system comprising 15 worker processes. It has been generated during the evaluation of a Mandelbrot graphics with 1000*1000 pixels. The rows of the profile are showing the activity of the processes over time. The worker processes inhabit the upper

IACSIT International Association of Computer Science and Information Technology
WWW.IACSIT.ORG

rows while the master activity is shown in the bottom bar. Light areas indicate high activity and dark areas indicate low activity.

Fig. 4: Activity profile of master-worker system with 15 workers

The profile clearly shows that the master is busy all the time while the workers are waiting for new tasks most of the time. This bottleneck in the master process typically occurs when the number of workers increases. The master process will not be able to keep all workers equally busy.

But a problem still exists. If the number of workers or submasters grows, the submasters also will be bottleneck because many communications appear between workers and their submasters. In the next section we propose a Linda-based architecture for hierarchical master-worker to decrease the communication cost.

III. The Linda Model

Linda is a parallel programming model for creation and coordination of multiple processes that run in one or more processors. The Linda model is embedded in a computation language (C, Lisp, etc.) and the result is a parallel programming language [11, 12].

The tuple space is a logical associative shared memory, a repository of elementary data structures, accessible only through the four Linda operations. A tuple is simply a sequence of values corresponding to typed fields. Linda provides operators for dropping tuples into the tuple space, removing tuples out of the tuple space and reading them without removing them. Associative search is used to find tuples in the tuple space. Templates, including values of a subset of the fields of a tuple, are used to select tuples for removal or reading. The Linda model defines four operations on the tuple space. These are:

out(t): it causes tuple t to be added into the tuple space.

in(s): it causes an arbitrary tuple t that matches the template s to be withdrawn from the tuple space. If such tuple does not exist, the call blocks.

rd(s): it is the same as in(s) expect that the matching tuple is not withdrawn from the tuple space.

eval(t): it causes a process to be created to evaluate the fields of the tuple t. When the evaluation ends the tuple t is put in the tuple space. Since the native environment already offers process creation, this operation was not implemented.

IV. Linda-based Hierarchical master-worker Architecture

In this section we describe our proposed architecture (Fig. 4) for resolving the problems that were mentioned in previous sections. Linda-based submasters provide a shared space (tuple-space) to save the tasks of the master for the workers and their results for the master.

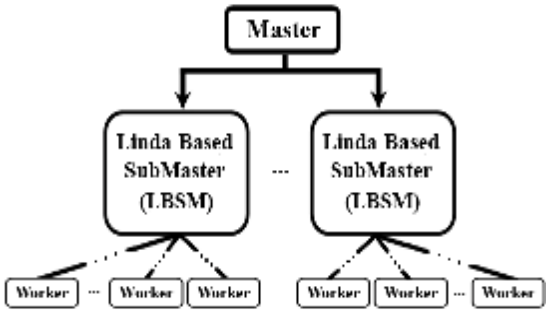


Fig. 5: Linda-based hierarchical master-worker architecture

Here, the workers can easily refer to their own submasters, by using the simple Linda operations, to take a task and put the results in/to the space. For large number of workers, if we use traditional hierarchical master-worker system, submasters will be under high workload. In this situation, a long queue of workers is created. They want to get a task from a submaster or give their result to it. On the other hand, if we have several levels of submasters, the communication between them is a critical problem.

We found the solution in assuming each submaster as a shared space in which each worker can easily access it, get a task and give back the results. We implement this shared space with Linad tuple spaces because it is easy to use and has simple operations. Therefore, each Linda-based submaster (LBSM) acts as a Linda tuple space.

In the next section we will report the effectiveness of our proposed architecture.

V. Experimental results

For evaluating the effectiveness of our proposed architecture, we execute a task on a cluster with 9 nodes. The primary task is multiplication of two large matrices. We test this experiment for four cases. At first, we execute this case study on a node with one processor. Then we test this experiment for common master-worker scheme with one master node and eight worker nodes. Finally, hierarchical and our Linda-based hierarchical models are examined with a structure which is shown in Fig. 6. P0 is the master, P1 and P5 are the submasters and P2, P3, P4, P6, P7, and P8 are the workers.

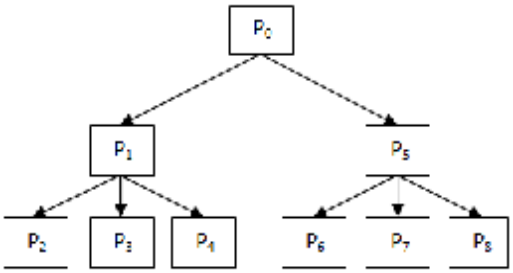
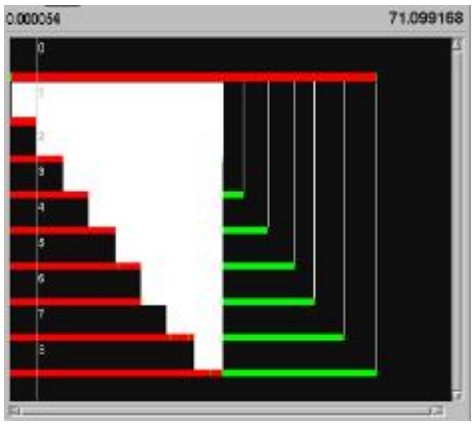


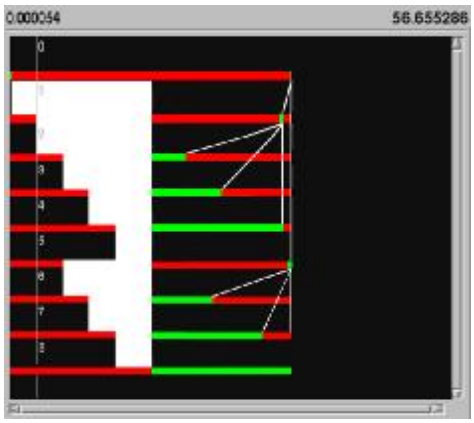
Fig. 6: Structure of the hierarchical and Linda-based master-worker systems in the experiment

The activity profiles for mentioned experiments are shown in Fig. 7. Green areas indicate high activity and red areas indicate low activity. White areas indicate wait on communication (if the paper is printed in color).

(a) Single processor



(b) Common master-worker system



(c) Hierarchical master-worker

(d) Linda-based hierarchical master-worker

Fig.7: The activity profiles of the experiment for different architectures

As shown in Fig. 7, our proposed system performed the task in shorter time. It happens because we eliminate some of the communication overheads. When the master put a task on the shared space placed in the submasters, all of their workers can access the tuple space concurrently. As a result the blocks of communication (white area) in Fig. 7.d are decreased noticeably. As soon as a submaster receives a result from a worker, the master can take it. On the other hand if the master put a task on submaster shared space, the workers can take it immediately.

Also, the workload on submaster is decreased. This experiment is done with small number of workers. Even though the number of workers is increased, the efficiency of the proposed architecture has not noticeable change. The reason is that we implement each submaster as a shared space.

When evaluating a parallel system, we are often interested in knowing how much performance gain is achieved by parallelizing a given application over a sequential implementation. Speedup is a measure that captures the relative benefit of solving a problem in parallel. It is defined as the ratio of the time taken to solve a problem on a single processing element to the time required to solve the same problem on a parallel computer with p identical processing elements [13]. We denote speedup by the symbol S. Therefore, speedup is defined as:

$$S(p) = \frac{Time(1)}{Time(p)} \qquad (1)$$

where Time(1) is the time taken to solve a problem on a single processing element and Time(p) is the time required to solve the same problem on a parallel computer with p identical processing elements.

Only an ideal parallel system containing p processing elements can deliver a speedup equal to p. In practice, ideal behavior is not achieved because while executing a parallel algorithm, the processing elements cannot devote 100% of their time to the computations of the algorithm.

Another parameter for evaluating a parallel system is efficiency. Efficiency is a measure of the fraction of time for which a processing element is usefully employed; it is defined as the ratio of speedup to the number of processing

IACSIT International Association of Computer Science and Information Technology WWW.IACSIT.ORG

elements [13]. In an ideal parallel system, speedup is equal to p and efficiency is equal to one. In practice, speedup is less than p and efficiency is between zero and one, depending on the effectiveness with which the processing elements are utilized. We denote efficiency by the symbol E. Mathematically, it is given by

$$E = \frac{S}{p} \quad (2)$$

where S is speedup and p is the number of processing elements.

We evaluate three master-worker architectures which are introduced in this section with execution time and two parameters speedup and efficiency. The task is multiplication of two large matrices. The results are summarized in Table 1, Table 2, and Table 3, respectively.

Table 1: Evaluation of common master-worker system with one master and 8 workers

No. of nodes	1	3	5	7	9
Speedup	1	2.44	3.88	4.49	5.51
Efficiency	1	0.81	0.78	0.64	0.61
Exec. Time (sec)	391.91	160.67	100.92	87.23	71.09

Table 2: Evaluation of hierarchical master-worker system with two submasters and 6 workers

Speedup	Efficiency	Exec. Time (sec)
6.91	0.77	56.65

Table 3: Evaluation of Linda-based hierarchical master-worker system with two submasters and 6 workers

Speedup	Efficiency	Exec. Time (sec)
8.05	0.89	48.65

As we see our proposed Linda-based hierarchical architecture shows better results in comparison to other approaches.

VI. Conclusions

Master-worker is a high-level programming framework that has been proposed to simplify the development of large parallel applications for Computational Grids. A common problem in traditional master-worker system is that the master is responsible for giving the tasks to the workers and gathering the results. Therefore, the master is bottleneck. For solving this problem a hierarchical master-worker model was presented. But with large number of workers, the submasters also become a bottleneck.

We proposed architecture in which each submaster as a shared space in which each worker can easily access it, get a task and give the result. We implement this shared space with Linad tuple spaces because it is easy to use and has simple operations. Therefore, each Linda-based submaster (LBSM) acts as a Linda tuple space. Evaluation of the proposed method showed the superiority of it in practice.

References

[1] Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infraestructure", Morgan-Kaufmann, 1999.
[2] J. Pierre G. J. Linderoth, M. Yoder "Metacomputing And the Master-Worker Paradigm". ANL/MCS-P792-0200, Mathemathics and Computer Science Division, Argonne National Labroratory, 2000.
[3] J, Berthold, M. Dieterle, R. Loogen, S. Priebe "Hierarchical Master-Worker Skeletons". LNCS 4902. Vol. 4902, pp. 248-264, 2008.
[4] J. B. Andrews, and C. D. Polychronopoulos, "An Analytical Approach to Performance/Cost Modeling of Parallel Computers." Journal of Parallel and Distributed Computing 12(4): 343-56, 1991.
[5] J. E. Hickman "An Analysis of an Interrupt-Driven Implementation of the Master-Worker Model with Application-Specific". Master Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University, 2007.
[6] M. A. Atkinson and Vishv "Coalescing idle workstations as a multiprocessor system using javaspaces and java web start". Internet and multimedia systems and applications, Kauai, Hawai, USA, IASTED International Conferance, Vol. 18, pp. 233-238, 2004.
[7] W. Glankwamdee, J. T. Linderoth MW: "A Software Framework for Combinatorial Optimization on Computational Grids", 2005.
[8] D. Abramson, J. Giddy, and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid", in Proc. of IPPS/SPDP, 2000.
[9] J.-P. Goux, S. Kulkarni, J. Linderoth, M. Yoder, "An enabling framework for master-worker applications on the computational grid", Tech. Report, University of Wisconsin – Madison, March, 2000.
[10] L. M. Silva and R. Buyya, "Parallel programming models and paradigms", in R. Buyya (ed.), "High Performance Cluster Computing: Architectures and Systems: Volume 2", Prentice Hall PTR, NJ, USA, 1999.
[11] S. Ahuja, N. Carriero, and D. Gelernter. Linda and Friends. IEEE Computer, 18:26-34,August 1986.
[12] N. Carriero and D. Gelernter. Linda in context. Communications of the ACM, 32, Number 4:444{558, April 1989.
[13] A. Grama, A. Gupta, G. Karypis, V. Kumar, "Introduction to parallel Computing", Addison Wesley, 2003.

Mohammad GhasemiGol was born in Birjand, Iran, in 1984. He received the B.S. degree in Computer Engineering from Payame Noor University (PNU), Birjand, Iran, in 2006. He is currently an M.S. student in Computer Engineering at Ferdowsi University of Mashhad (FUM), Iran. He is a member of Young Iranian Elites Association and High Performance Computing Group of FUM. His research interests include grid computing, distributed systems, parallel algorithms and parallel programming, distributed shared memory systems, machine learning, intelligent data mining, intrusion detection, and optimization.

Mostafa Sabzekar was born in Birjand, Iran, in 1985. He received the B.S. degree in Computer Engineering from Tarbiat Moallem University of Tehran, Iran, in 2007. He is currently an M.S. student in Computer Engineering at Ferdowsi University of Mashhad, Iran. His research interests include knowledge-based system, machine learning, data mining, and optimization.

Hossein Deldari received his B.Sc. degree in Physics from University of Mashhad, Iran in 1976. He received his Masters degree in Computer science from University of Oregon, EUgene, Oregon, USA, in 1979 and joined the Department of Computer Engineering at Ferdowsi University of Mashhad. He received his Ph.D. in Parallel and distributed systems from University of Leeds, Leeds, England,

in 1995. He has been involved in the research and development of parallel and distributed systems for almost a decade. His research interests include parallel algorithmic skeletons, parallel fuzzy genetic algorithms, and grid / cluster computing.

Amir-Hassan Bahmani was born in 1986. He received the B.S. degree in Computer Engineering from Islamic Azad University of Mashhad (IAUM), Mashhad, Iran, in 2009. He is a member of Young Iranian Elites Association and IAENG. His research interests include parallel and distributed algorithms, database management systems, software development methodologies.

IACSIT
International Association of Computer Science and Information Technology
WWW.IACSIT.ORG