Research and implementation of the heterogeneous earliest finish time algorithm based on Pi-Calculus

Hui Kang[1, a], Huiping Fei[1, b] and Fang Mei[1, *]

[1]School of Computer, JiLin University, Changchun 130012, China;

[a]kanghui@jlu.edu.cn, [b]feihp14@mails.jlu.edu.cn, *Corresponding author Email:meifang@jlu.edu.cn

Keywords: Task scheduling algorithm, HEFT, pi-calculus, PICT.

Abstract. Task scheduling algorithm in heterogeneous environment is programmed in a serial manner in the underlying by using traditional programming language. In this paper, we proposed that using the characteristics of parallel computation of Pi-calculus, to parallel analysis and modeling of the heterogeneous earliest finish time (HEFT) algorithm based on pi-calculus. Then programmed it using PICT, which is a programming language that corresponds to the pi-calculus. By creating different topologies of the task scheduling graph and compared with traditional programming language (C++ language) prove that the efficiency of using pi-calculus to develop the HEFT algorithm is higher than that of traditional programming language.

1. Introduction

Multi-task scheduling in a distributed environment was demonstrated to be NP-complete [1]. Researchers are still continuing with attempts to improve the algorithm. The HEFT algorithm, which was proposed by Haluk Topcuoglu et al., is a classic static scheduling algorithm in a heterogeneous environment. The HEFT algorithm includes two main steps. The first step involves computing the upward rank value of each task and determining the priority of the task based on the rank upward value. The second task involves computing the processor. In this step, the processor that can complete each task at the earliest time is selected based on an insertion-based policy and according to the priority order of the tasks, as shown in Figure 2.

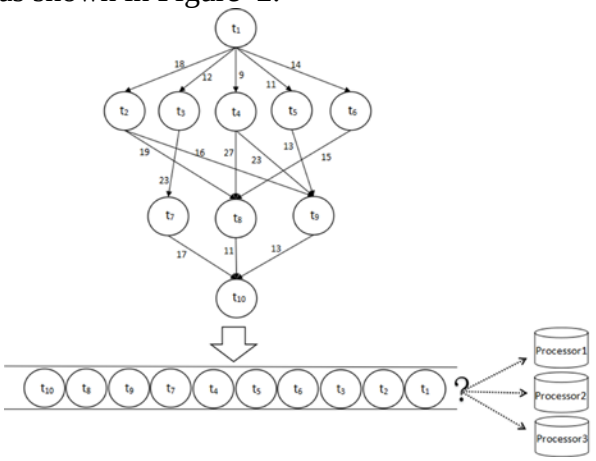

Fig. 2 HEFT algorithm schematic

2. Modeling the heterogeneous earliest finish time algorithm based on Pi-calculus

2.1 Pi-calculus

Pi-calculus [4] was developed by Robin Milner, it is a calculation model that describes and analyzes concurrent systems, and extends the CCS and communication based on a channel. This model allows the channel to transfer data along other channels. An extension introduces element mobility through the dynamic evolution of the communication topology to regulate and test concurrent systems. Channel mobility increases the strength of the expression ability of pi-calculus

but maintains simple semantics and easy algebraic theory. We employ the functional programming with lists [5] of pi-calculus.

Definition 2.1 Lists The constant Nil, construction $Con(V, L)$ and a list of n values are defined as follows:

$$Nil \overset{def}{=} (k).!k(nc).\bar{n}$$

$$Cons(V, L) \overset{def}{=} (k).new\ \ vl(Node < kvl > | V < v > | L < l >)$$

where

$$Node(kvl) \overset{def}{=} k(nc).\bar{c} < vl >$$

$$[V_1, \cdots, V_n] \overset{def}{=} Cons(V_1, Cons(\cdots Cons(V_n, Nil)\cdots))$$

Note that the values are defined to have no free names where

L<k>: represents place a copy of a value L at some location k;

V: indicates data type of data stored in the list; and

L: denotes the list as defined in definition 2.1 on this kind of type data.

Definition 2.2 Operation of read the element in L<k>:

$$\bar{k} < nc > .c(vl).P \,|\, Cons(V, L) < k >$$

where

v: Interface for interactive information at element

l: Interface for interactive information at table footers

Figure 3 presents the task scheduling graph list. Each "Node" represents a task, and "struct" is a type of self-defined data. The node ID, average computing time of the node, child nodes list, and parent nodes list are included.

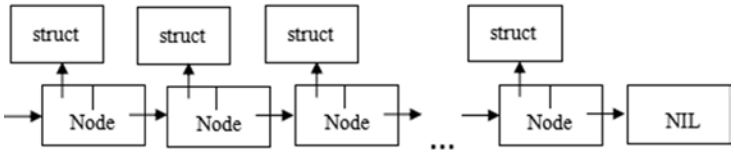

Fig. 3 Task scheduling graph list

The data structure required in this paper is similar to the task scheduling graph list, although the difference is "struct".

2.2 Modeling

We model the HEFT algorithm based on pi-calculus. First, the upward rank value of each node is computed.

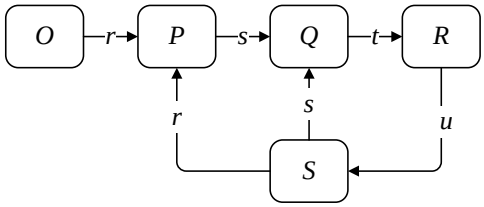

Fig. 4 Channel graph between the processes of the first stage

$$O = \bar{r} < l, rank > | P \tag{1}$$

l: the task scheduling graph list

r: upward rank value

$$P = r(link, rank).\overline{link} < nc > .c(vl).(if\ (v.next\ is\ null) \Rightarrow setrank(v.id, v.comp, rank)$$
$$else \Rightarrow \bar{s} < v.next, v.id, l, rank > | Q) \,|\, Cons(V, L) < link > \tag{2}$$

setrank (id, value, rank): according node id to find the rank value of the task in rank list. The original value is assessed to determine whether it is less than value if it is replaced by the value.

$$Q = s(childlink, id, link, rank).\overline{childlink} < nc > .c(vl).\bar{t} < v, childlink, id, link, rank > | R \\ | Cons(V, L) < childlink > \tag{3}$$

$$R = t(childnode, childlink, id, link).setrank(id, getrank(childnode.id) + childnode.comm). \\ \bar{u} < childlink, link > | S \tag{4}$$

$$S = u(childlink, link, rank).if\ (childlink\ is\ null) \Rightarrow if\ (link\ is\ null) \\ \Rightarrow sort(rank).\overline{step2} < rank > else \Rightarrow \bar{r} < link > | Pelse \Rightarrow \bar{s} < childlink, id, link > | Q \tag{5}$$

Equation (1): process O sends the task scheduling graph list l and upward rank value list rank to process P along channel r.

Equation (2): process P receives l, and a node is taken out from l. The children list of the node is empty if the upward rank value of the node is the average computation time of the node. Otherwise, send the children list and id of the node to process along channel s.

Equation (3): process Q receives the children list from process P, takes out a node of the list, and sends it to process R along channel t.

Equation (4): process R receives the child node from process Q, and the upward rank value of the child node and the average communication time with the child node are obtained. The setrank function is called, and the rank value modified. The remaining parts of the children list are sent to process S along channel u.

Equation (5): process S determines whether the remaining child node is empty. If the remaining child node is empty, the upward rank value is calculated and the remaining list of the task scheduling graph is sent to process P along channel r. Otherwise, the remaining children list is sent to process Q along channels.

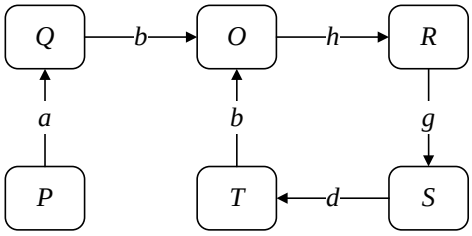

Fig. 5 Channel graph between processes of second stag

Figure 5 is the Channel graph of second task, as a result of similarity with first task so omitted. PICT language was designed and implemented by Benjamin C. Pierce and David N. Turner from the Computer Science Department of Edinburgh University. This language was designed to implement a high-level concurrent language in terms of the pi-calculus primitives. PICT has its own unique manner of processing, and is very close to the pi-calculus ideas. The variable in PICT cannot be changed. Thus, no multiple processes can modify one variable at the same time, which leads to a "deadlock" problem that reduces the cost of the concurrency control.

3. Comparison of experimental results

In the experiment, the differences between the topology structures in the task scheduling graph are considered. For reliability, the experiment is divided into a condition with relatively large longitudinal and lateral depths. For each situation, we set the number of nodes to 10, 20, 30, 40, and 50, as an example.

The experiment environment for this article is a centos system under VMware virtual machine, VMware 6.3, which has a memory of 628 M, and a hard disk is 20 G.

The experiment uses PICT and C++ programming languages to implement the HEFT algorithm. Time is adopted as the measurement function in the Linux system to obtain the three types running time: real, user, and sys time, which represent the actual running time of the program, running time in user mode, and running time under the system state mode, respectively.

3.1 Longitudinal depth is relatively large

Table 2 Comparison of real time in the case of Longitudinal depth is relatively large (ms)

Number of nodes	PICT	C++
10	160.6	664.6
20	190.9	687.4
30	215	713.6
40	247.2	692.3
50	326.9	792.8

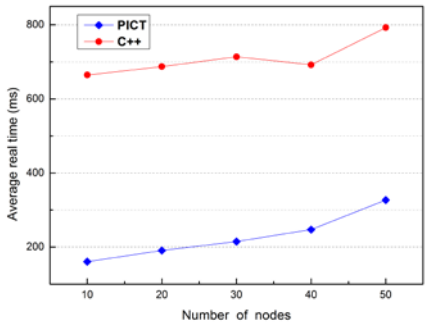

Fig. 6 Comparison of real time in the case of Longitudinal depth is relatively large (ms)

Table 2 and Figure 6 indicate real time change when the number of the nodes is 10, 20, 30, 40, and 50 in the case of a larger vertical depth. The figure shows that the running time of PICT is 500 ms faster than the running time of the C++ language. The advantage is maintained when the number of nodes increases. Thus, with the increase of the number of the nodes, the efficiency of PICT is still higher than C++.

Table 3 Comparison of user time in the case of Longitudinal depth is relatively large (ms)

Number of nodes	PICT	C++
10	117.4	443.1
20	148.9	431.8
30	170.3	439.7
40	203.5	432.3
50	257.7	439.8

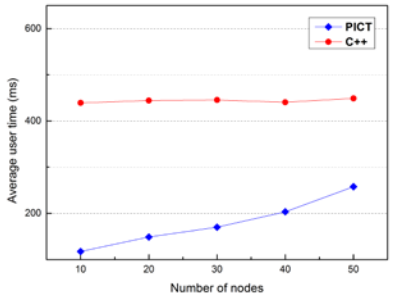

Fig. 7 Comparison of user time in the case of Longitudinal depth is relatively large (ms)

Table 3 and Figure 7 present of the running time in user mode on the realization of HEFT algorithm in the PICT and C++ languages in the case of larger lateral depth. The figure shows that the efficiency of PICT remains higher than that of the C++ language.

Table 4 Comparison of sys time in the case of Longitudinal depth is relatively large (ms)

Number of nodes	PICT	C++
10	17.1	141.4
20	15.9	143.1
30	19.4	129.7
40	18.4	136.8
50	24.3	156.8

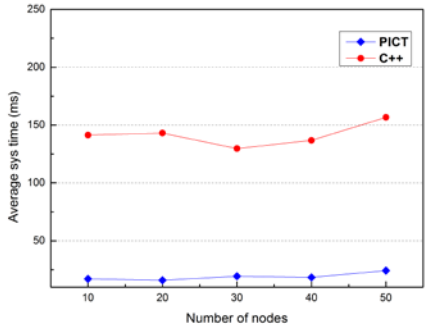

Fig. 8 Comparison of sys time in the case of Longitudinal depth is relatively large (ms)

Table 4 and Figure 8 present the system time in the case of larger lateral depth. The figure shows that the efficiency of PICT remains higher than that of the C++ language. PICT is more stable than the C++ language, which demonstrates the advantages of PICT.

3.2 Lateral depth is relatively large

Table 5 Comparison of real time in the case of lateral depth is relatively large (ms)

Number of nodes	PICT	C++
10	182.6	719.7
20	217.5	803.7
30	242.8	808.2
40	282.8	806.9
50	316.1	839.4

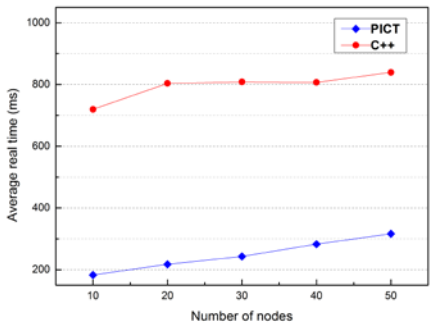

Fig. 9 Comparison of real time in the case of lateral depth is relatively large (ms)

Table 5 and Figure 9 indicate the running time in real time on the realization of the HEFT algorithm in PICT and C++ language in the case of the larger vertical depth. Figure 5.4 shows that PICT is still faster than the C++ language despite the number of the nodes being more than 50.

Table 6 Comparison of user time in the case of lateral depth is relatively large (ms)

Number of nodes	PICT	C++
10	182.6	719.7
20	217.5	803.7
30	242.8	808.2
40	282.8	806.9
50	316.1	839.4

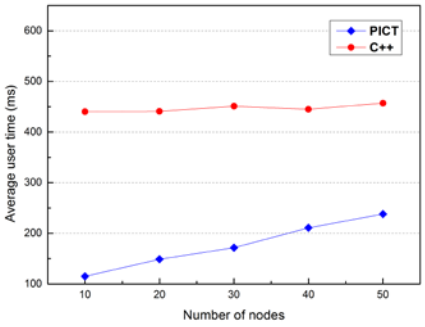

Fig. 10 Comparison of user time in the case of lateral depth is relatively large (ms)

Table 6 and Figure 10 indicate the running time in user mode on the realization of HEFT algorithm in PICT and C++ language in the case of larger vertical depth. The figure shows that the efficiency of PICT remains higher than that of the C++ language.

Table 7 Comparison of sys time in the case of lateral depth is relatively large (ms)

Number of nodes	PICT	C++
10	21.9	152.1
20	20.5	159
30	22.6	153.2
40	23.8	158.2
50	25.9	162

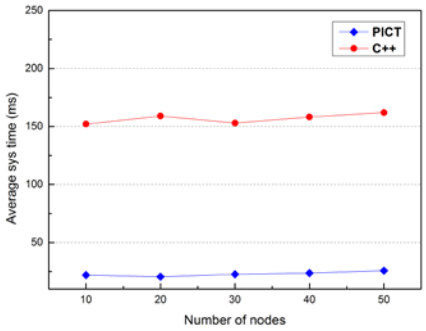

Fig. 11 Comparison of sys time in the case of lateral depth is relatively large (ms)

Table 7 and Figure 11 show the running time in the system state mode on the realization of HEFT algorithm in the PICT and C++ language in the case of larger vertical depth. The growth trend of the two languages is moderate, and the efficiency of PICT remains higher than that of the C++ language.

4. Conclusion

The HEFT algorithm is a classical heterogeneous scheduling algorithm proposed many years ago. Since its proposal, a number of researchers have improved and optimized this algorithm. However, pi-calculus has never been adopted to describe and implement the algorithm. In the current paper, the heterogeneous static scheduling algorithm and pi-calculus were studied and analyzed. A model of the HEFT algorithm based on Pi-calculus is also presented and implemented by adopting the PICT language. The experimental results show that the distributed concurrent pi-calculus is more efficient than the traditional high-level programming language (C++ language) in obtaining the HEFT algorithm.

References

[1] Ullman J D. NP-complete scheduling problems. Journal of Computer and System Sciences, 1975, 10(3): 384-393J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

[2] Haluk Topcuoglu, Salim Hariri, Min-You Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE transactions on parallel and distributed systems. 2002

[3] Georgios L.Stavrinides, Helen D.Karatza:Scheduling multiple task graphs with end-to-end deadlines in distributed real-timesystems utilizing imprecise computations. The Journal of Systems and Software.2010.

[4] Milner, Robin. Functions as Processes.In Research Report 1154, INRIA, SophiaAn-tipolis. Final version. in J. Mathem. Struct. In Computer Science. 1990.

[5] Milner Robin. Communicating and Mobile Systems: The Pi-Calculus, Cambridge University Press, Cambridge, UK, 1999.

[6] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, Proof, Language and Interaction: Essays in Honour of Robin Milner, Foundations of Computing. MIT Press, May 2000.

[7] Benjamin Pierce and David Turner . Pict language definition. Draft repor; available electronically as part of the Pict distribution，1997

[8] Benjamin C.Pierce, Davide Sangiorgi. Behavioral Equivalence in the Polymorphic Pi-Calculus[J]. Journal of the ACM, 2000, Volume 47, Issue 3, pp.531-584.

[9] Pawel T. Wojciechowski. Nomadic Pict Language Libraries Release 1.0-alpha. 2000

[10] Pawel T. Wojciechowski. The Nomadic Pict System Release 1.0-alpha Documentation and user's manual. 2000

[11] WANG Xiao-Le, HUANG Hong-Bin, DENG Su. List Scheduling Algorithm for Static Task with Precedence Constraints for Cyber-physical Systems. acta automatica sinica. 2012

[12] KANG Hui, ZENG Ying-ying, LIU Zhi-yong. Modeling the mobile communication service based on PI-calculus. Journal on Communications. 2009.