Proceedings of the 8th WSEAS Int. Conference on Automatic Control, Modeling and Simulation, Prague, Czech Republic, March 12-14, 2006 (pp255-260)

Evaluation of Parallel Processing Systems with Erlang Concurrent Traffics Using Average Concurrency

CHANINTORN JITTAWIRIYANUKOON
Department of Telecommunications Science
Faculty of Science & Technology
Assumption University
Ramkhamhaeng 24, Hua Mak, Bangkok
THAILAND

Abstract: - This paper presents a queueing model to measure the performance of parallel processing systems (PPS) with Erlang distribution of concurrent traffics. Several methods have been proposed such as decomposition technique, which is applicable to only fixed concurrent traffics. On the other hand, this paper will propose an innovative approach considering a more practical set of concurrent traffics. They are both coming to the PPS and taking service in the PPS with the Erlang distribution. To avoid both the state space explosion and the calculation complexity, as practical concurrent traffics exist, the average concurrency (AVC) is determined which is the average of the overall concurrent traffics. The results apparently indicate that this proposed method can reduce the number of states from the exponential increment of power of N to the increment of power of average number of concurrent traffics. This results a faster computational achievement. For various examples, the before-and-after applying AVC method is compared by simulations. In addition, the computational cost is discussed.

Key words: - Parallel processing, granularity, parallelism, queueing model and performance evaluation.

1 Introduction

There have been several attempts to evaluate the performance of parallel processing systems (PPS) with a set of concurrent traffics. Some researchers treated as fixed concurrent traffic in order to simplify the estimation for the system as such [1]-[7]. Some may evaluate the concurrent traffics by using simulation. In [3], an analysis method called decomposition approximation was proposed. The authors proposed the analysis method based upon mean value algorithm (MVA) [8]. Although the high accuracy of the method has been discussed but the technique proposed in that paper is applicable only to an unchangeable set of concurrent traffics. In practice, the system is shared by users, as in time-sharing systems, the concurrency of individual programs is never identical.

Considering the realistic situation [10], however, where the computer is almost personalized, as in the case where a workstation is shared by a small number of users over the ATM network, the detailed characteristic of individual programs must be taken into account in the analysis.

Our analysis is based upon the average concurrent traffics. This technique will allow us to derive a two-step simpler calculation by decomposing the average concurrency into upper and lower bounds. The interesting characteristic parameters such as utilization, throughput and mean queue length can be obtained in two sets separately, the lower and the higher bound sets. The two sets are then averaged with weight, providing a final solution for the observed system.

Our results focus on an avoidance of the state space explosion and calculation complexity, results between before-and-after applying proposed AVC technique will be compared by simulation. As computer networks were invented, the number of applications with parallelism that can be executed by the distributed networks have been increasing. Ongoing researches will focus on partitioning process to improve the parallel processing by making it faster and more reliable. The application and software designers cannot emphasize only on the data exchange, they must find a better way to increase the concurrent computing power [8]. Research on parallel and distributed system on computer network may provide the answer [9]. For the last decade, the computers evolution has been changed drastically both software and hardware. On

the hardware side, the development is concentrated on "Multi-processor", a scheme which more than one CPUs are loosely working together to perform an assigned task. This kind of development is expected to continue in a foreseeable future. Although it is not cost-effective to implement, the multiprocessor scheme will help improve the efficiency of a computation. Another alternative is to use PPS (a system utilizes a number of processors to compute subtasks simultaneously) to shorten the overall processing time. This paper will focus on PPS to improve the processing speed with a practical concurrent task. In practice, the subtask programs after partition can be identical or different in size. Once individual subtask processing is completed then it must wait for others until the last subtask from the same original task has been completed. All subtask results will then be merged to mark the completion of one task by PPS.

2 Analytical Model

This section presents the analytical model used in this paper. First, it is assumed that the total programs composing of N completely independent, individual programs. Each of those programs will be called the primary task. The PPS contains several parallel processing units and one pseudo-server. The primary task will enter the pseudo-server with no service time prior to the PPS entry. In this pseudo-server, we allow primary tasks to split into several secondary tasks. The secondary task spawned by the same primary task is called sibling. All siblings is subsequently processed by PPS. It is assumed that there is no data dependency among siblings, except for the effect of queue in front of service unit in the PPS. It is assumed that the service time at each visit will follow the Erlang distribution. When each sibling completes the execution in the PPS, it has to return to the buffer of the pseudo-server. If the execution in another siblings is yet not completed, the finished sibling has to wait for their completion in the buffer. When all sibling completion is achieved, they will be merged into the primary task. At this point, the synchronization of siblings starts and the process is repeated. The duration from when primary task enters a pseudo-server to when it returns is called a cycle time. If a primary task (with identifier i) is divided in a pseudo-server into X_i secondary tasks then it is defined that the secondary tasks have the concurrency X_i . In general, X_i changes cyclically. The concurrency for each cycle is represented by a vector $C_i = \{X_{i1}, X_{i2}, X_{i3}, \ldots\}$ called the concurrency vector.

3 Average Concurrency Computation

This section will focus on evaluation of the system performance if **DCP** (Different Concurrent Programs), **CCP** (Cycle-Dependent Concurrent Programs) or **VCP** (Variable Concurrent Programs) is executed.

The average concurrency (AVC) of the overall programs $\overline{C}$ is calculated. Using the integers $\overline{C}_1$ and $\overline{C}_2$ ($\overline{C}_2 = \overline{C}_1 + 1$) bounding $\overline{C}$ and, using the EZSIM [6] simulation to collect characteristic parameters (such as throughput, mean queue length, utilization etc.) The two sets are then calculated with weight, giving final results for the observed system as shown in figure 1.

The following symbols are defined.

N : number of overall primary tasks

i : identifier of primary tasks (i = 1,2,3,…, N)

C_i : concurrent vector of primary task i

M_i : number of elements in C_i

X_{ij} : concurrency at the j-th cycle of the primary task I

$\overline{C}$: average concurrency

$\overline{C}$ is calculated as follows for DCP, CCP and VCP.

DCP

$$Ci = \{Xi_1\} \text{ for } i = 1,2,3,\ldots,N$$

$$\overline{C} = \frac{1}{N}\sum_{i=1}^{N} X_{i1}$$

CCP

$$Ci = \{X_{11}, X_{12}, X_{13,\ldots,} X_{1M_1}\} \text{ for } i = 1,2,3,\ldots,N$$

$$\overline{C} = \frac{1}{M_1}\sum_{j=1}^{M_1} X_{1j}$$

VCP

$Ci = \{X_{i1}, X_{i2,...,} X_{iMi}\}$ for $i = 1,2,3,\ldots,N$

$$\overline{C} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{Mi} Xij}{\sum_{i=1}^{N} Mi}$$

/********** **ALGORITHM** **********/

BEGIN /*** Initialize $\overline{C}_1$ and $\overline{C}_2$ ***/
/*** I is integral part of $\overline{C}$ ***/
/*** F is fractional part of $\overline{C}$ ***/
INITIALIZE $\overline{C}_1$ = I
$\overline{C}_2$ = I + 1
END

BEGIN/**** Collect performance parameters as concurrent level is $\overline{C}_1$ and $\overline{C}_2$ ****/
FOR n = $\overline{C}_1$ TO $\overline{C}_2$ **DO {**
Collect mean queue length MQL_n
Collect utilization factor $UTIL_n$
Collect throughput $THRU_n$ **}**
END FOR
END

BEGIN /*** Final calculation ***/
MQL($\overline{C}$)=(1-F)MQL($\overline{C}_1$) + (F)MQL($\overline{C}_2$)
UTIL($\overline{C}$)=(1-F)UTIL($\overline{C}_1$) + (F)UTIL($\overline{C}_2$)
THRU($\overline{C}$)=(1-F)THRU($\overline{C}_1$) + (F)THRU($\overline{C}_2$)
END

/****** **END OF ALGORITHM** *****/

Figure 1. Algorithm of Average Concurrency Method.

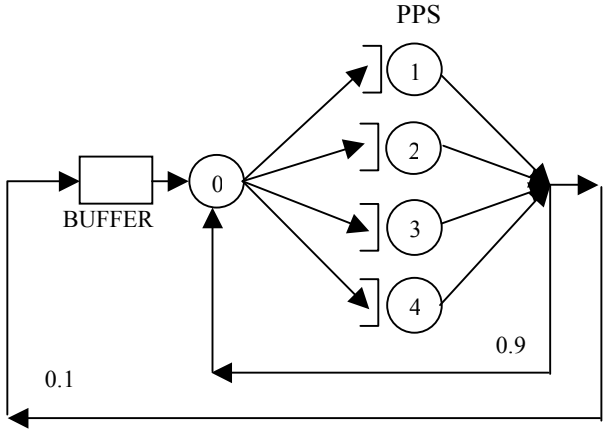

Figure 2. PPS with 4 Processing Units.

4 Simulation Model

We label these processing units as server 0-4. Server 0 is a pseudoserver with no service time. The queue discipline is set to be first-in first-out (FIFO) except for server 0. The server 1-4 will be an interconnection of servers with the same branching probability. The four servers 1-4, are connected in parallel. These service time distributions are Erlangian distribution stage n with mean 0.01 second. Erlang distribution is shown in equation 1.

$$E_n(x) = \frac{\mu^n x^{n-1} e^{-\mu x}}{(n-1)!}. \qquad (1)$$

where x=1,2,3…and n is a positive integer stage number.

A PPS with four processing units is considered as shown in figure 2. Another examples of PPS can be found in [10],[11],[12]. In the simulation, μ will be set to 0.01 second and n will be set to two. To investigate validation of the proposed AVC method two examples from each of DCP, CCP and VCP are used as follows.

DCP1 : N=4, C_1={1}, C_2 ={2}, C_3={3}, C_4 ={4}

DCP2 : N=5, C_1={2}, C_2 ={3}, C_3={4}, C_4 ={5}, C_5 ={6}

CCP1 : N=4, C_1=C_2 =C_3=C_4 ={2,3,4}

CCP2 : N=5, C_1=C_2 =C_3=C_4 = C_5 ={3,4,5}

VCP1 : N=4, C_1={1,2,3}, C_2 ={1,1,2}, C_3={2,3,4}, C_4 ={3,3,4}

VCP2 : N=5, C_1={1,3,5}, C_2 ={2,3,5}, C_3={1,1,4}, C_4 ={2,2,3}, C_5 ={1,2,3}

5 Results and Analysis

To validate the average concurrency method, we firstly consider the number of states used in the computation then the average concurrency method will reduce state number size from $(2^{\overline{C}})^N$ to $(N+2)(N+1)^{\overline{C}_1}$. As an example, state number used by straightforward (**STRF**) method and by average concurrency method is compared and listed in table 1.

DCP	STATES BY **STRF**	STATES BY **AVC METHOD**	REDUCTION PERCENTAGE
X_{11}=2 X_{21}=3 X_{31}=4	315	54	82.85%
X_{11}=2 X_{21}=3 X_{31}=3	147	64	56.46%
X_{11}=1 X_{21}=1 X_{31}=3 X_{41}=3	49	15	69.39%
X_{11}=2 X_{21}=2 X_{31}=4 X_{41}=4	2025	105	94.81%
X_{11}=1 X_{21}=2 X_{31}=3 X_{41}=4	315	120	61.90%
X_{11}=2 X_{21}=4 X_{31}=4 X_{41}=4	10125	695	93.14%

Table 1. State Space Comparison.

DCP			
CASE 1	**PPS**	**STRF**	**AVC**
MEAN QUEUE LENGTH	1	0	0
	2	0	0
	3	0	0
	4	0	0
UTILI-ZATION	1	0.4 .	0.39
	2	0.4 .	0.39
	3	0.4 .	0.39
	4	0.4 .	0.39
THROUGHPUT	0	8	7.5
CASE 2	**PPS**	**STRF**	**AVC**
MEAN QUEUE LENGTH	1	0.003	0
	2	0.004	0
	3	0.0039	0
	4	0.004	0
UTILI-ZATION	1	0.49 .	0.5
	2	0.48 .	0.5
	3	0.49 .	0.5
	4	0.49 .	0.5
THROUGHPUT	0	4	4

Table 2. Comparison of Results versus STRF (DCP).

CCP			
CASE 1	**PPS**	**STRF**	**AVC**
MEAN QUEUE LENGTH	1	0	0
	2	0	0
	3	0	0
	4	0	0
UTILI-ZATION	1	0.43 .	0.45
	2	0.44 .	0.45
	3	0.42 .	0.45
	4	0.44 .	0.45
THROUGHPUT	0	6	6
CASE 2	**PPS**	**STRF**	**AVC**
MEAN QUEUE LENGTH	1	0.005	0
	2	0.004	0
	3	0.0039	0
	4	0.0049	0
UTILI-ZATION	1	0.46 .	0.5
	2	0.48 .	0.5
	3	0.49 .	0.5
	4	0.49 .	0.5
THROUGHPUT	0	3	4

Table 3. Comparison of Results versus STRF (CCP).

VCP			
CASE 1	**PPS**	**STRF**	**AVC**
MEAN QUEUE LENGTH	1	0	0
	2	0	0
	3	0	0
	4	0	0
UTILI-ZATION	1	0.38 .	0.4
	2	0.39 .	0.4
	3	0.4 .	0.4
	4	0.4 .	0.4
THROUGHPUT	0	8	7
CASE 2	**PPS**	**STRF**	**AVC**
MEAN QUEUE LENGTH	1	0.001	0
	2	0.001	0
	3	0	0
	4	0.001	0
UTILI-ZATION	1	0.37 .	0.39
	2	0.38 .	0.39
	3	0.39 .	0.39
	4	0.37 .	0.39
THROUGHPUT	0	8.5	7.5

Table 4. Comparison of Results versus STRF (VCP).

Results obtained by average concurrency (AVC) method are compared versus the straightforward (STRF) results. Tables 2-4 are results of DCP, CCP and VCP respectively. It can be apparently seen that mean queue length, servers utilization and throughput results collected from both STRF and AVC methods are close. It is because the variance of concurrent levels does not deviate from average concurrent level much. The examples also supported a practical aspect that primary programs have to spawn a small number of independent secondary tasks. It is the matter of fact that the tasks will need infrequent synchronization, which is considered as processing overhead. The overhead may become disastrous if concurrent level is higher than the available processing resources (four processing units in this experiment). Primary programs are recommended no partition and should obtain service in serial fashion whenever overhead becomes significant. Different examples with high variance of concurrent levels are under investigation.

6 Conclusion

The AVC method for parallel processing system with concurrent programs is introduced. The validation and accuracy of the AVC method was found by checking results against ones obtained by the straightforward (STRF) method. It can be clearly seen by table 2-4 that only few differences in estimating the servers utilization, throughput and mean queue length are listed. As can be seen, values obtained by the AVC method agree very favourable with values obtained by the STRF. It is reasonably concluded that the AVC method validates for DCP, CCP and VCP.

Regarding the maximum number of states to be calculated, the AVC method is compared with the STRF. The number of states in the AVC method increases in proportion to the power of the average concurrent level. On the other hand, the required number of states in the STRF increases exponentially. Since the computational cost is proportional to the cube of the number of states, there will be a great difference regarding the computational cost. Although the AVC requires that the calculation be done twice for lower and upper level as shown in figure 1, the estimation can be achieved with far less number of states and computational cost than the STRF. Finally, the AVC method can be executed on any systems with a small memory capacity, and can be executed with higher speed.

7 References

[1] Fujimoto and Richard M., Parallel and Distributed Simulation Systems., 2000.

[2] Sutaweesup, Wasara, and Yuen Poovorawan, Parallel Motion Path Calculation for Animated Objects in Distributed Environment, 2000.

[3] Kormicki, Maciek, Ausif Mahmood and Bradley S. Carlson, Parallel Logic Simulation on a Network of Workstations Using a Parallel Virtual Machine, Washington State University at Tri-Cities, University of Bridgeport, State University of New York at Stony Brook, pp. 123-134, 1997.

[4] Wail Elias Mardini, Modeling with D-queues, The University of New Brunswick, April, 2001.

[5] Donald Gross and Carl M. Harris, Fundamentals of Queueing Theory, third edition, Canada, A Wiley-Interscience Publication, 1998.

[6] Behrokh Khoshnevis, Discrete systems simulation, University of southern California, McGraw-Hill inc., 1994.

[7] P. Viriyaphol et. al., Evaluation of Routing Techniques Over IP-based Networks, Proc. IASTED International Conference on Internet and Multimedia Systems and Applications (EuroIMSA05), pp. 158-162,2005.

[8] Kai Hwang et. al, Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space, IEEE Concurrency, vol.7 (1), Jan.-March, 1999.

[9] K. Hwang and Z. Xu, Scalable Parallel Computing Technology, Architecture, Programming, WCB/McGraw-Hill, NY, 1998.

[10] Lufs Moura e Silva and Rajkumar Buyya, Parallel Programming Models and Paradigms, Monash University, Melbourne, Australia, 1999.

[11] Sinha and Alok, Client-Server Computing, ACM Transaction of Communications, pp. 220-231, 1992.

[12] McDonald Chris, and Kamaran Kazemi, Improving the PVM Teaching Environment, Department of Computer Science, The University of Western Australia, pp. 219-223, 1997.