Journal of Southeast University (English Edition) Vol. 21, No. 4, pp. 419 – 426 Dec. 2005 ISSN 1003—7985

Load balancing framework for actively replicated servers

Wang Yun Wang Junling

(Department of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

Abstract: This paper focuses on solving a problem of improving system robustness and the efficiency of a distributed system at the same time. Fault tolerance with active replication and load balancing techniques are used. The pros and cons of both techniques are analyzed, and a novel load balancing framework for fault tolerant systems with active replication is presented. Hierarchical architecture is described in detail. The framework can dynamically adjust fault tolerant groups and their memberships with respect to system loads. Three potential task scheduler group selection methods are proposed and simulation tests are made. Further analysis of test data is done and helpful observations for system design are also pointed out, including effects of task arrival intensity and task set size, relationship between total task execution time and single task execution time.

Key words: load balancing; fault tolerance; framework; task scheduler group

Computer software and hardware are not so robust that their mistakes may lead to system collapse. Nowadays, the demands of applications for better performance are stronger. Hence, how to enhance system robustness and high efficiency is a hot point.

Fault tolerance (FT) with active replication in asynchronous distributed systems is good at enhancing system robustness and improving reply performance[1]. Critical parts in a system are replicated and replicas comprise an FT group. The copies of the critical server run concurrently. All of them receive the requests from clients, execute the corresponding operations, and send back the responses. When a failure occurs on one replica detected by the failure detector, the failure is masked and the computation continues as long as there are enough operational replicas in a group. This replication style assumes a deterministic behavior on these replicas and requires an atomic broadcast mechanism to maintain consistency. If there are not too many replicas that fail simultaneously, a system is able to survive. The drawback of this technique is that due to replicas in a system, to a certain extent, resource waste exists in a system during fail-free periods.

Load balancing (LB) protects a server from overload. It is good at improving request reply performance while it is able to provide fault tolerance in certain situations, for example, migrating a request dynamically. Therefore, fault tolerance with active replication and load balancing techniques are complementary. The marriage of fault tolerance and load balancing in an asynchronous distributed environment is a good way to maintain high system performance and robustness at the same time. Such a technique is useful in the applications of Web server, weather forecast computing, etc.

The main contribution of this paper is two-fold. One is to propose a load balancing framework for fault tolerance with active replication. The framework is able to dynamically adjust FT groups and their memberships due to system loads. An FT group may change its redundancy due to members joining or leaving. The number of FT groups may also be increased or decreased due to heavy or light system task load. The framework is good at scalability and load adaptation. The other is to manifest the effects of task arrival intensity and task set size on performance, and the relationship between total task execution time and single task execution time.

1 Related Work

In asynchronous distributed systems, fault tolerance problems with active replication are deduced to a Consensus problem[2]. FLP impossibility in 1985, shows that it is impossible to distinguish a slow processor from a crashed processor in an asynchronous environment[3]. Chandra and Toueg pointed out that the Consensus problem is solvable in an asynchronous distributed environment if processors are equipped with failure detectors at least satisfying $\diamond S$ properties[2].

Received 2005-06-06.

Foundation items: The National Natural Science Foundation of China (No. 60273038); the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry; Program for New Century Excellent Talents in University, MOE (No. NCEF-04-0478).

Biography: Wang Yun (1967—), female, doctor, professor, yunwang@seu.edu.cn.

The Chandra-Toueg Consensus protocol[2] shows the steps and details to solve the Consensus problem theoretically. There are also some fault tolerant systems, such as the SIFT system[4], the State Machine[5], the Isis (and later Horus) system[6,7], the Psync system[8], etc. These systems are not able to adjust the systems in terms of requested task loads. Computation resources in a system are not fully used in the case of fail-free.

Research on load balancing mainly focuses on load balancing strategies, such as static and dynamic load balancing strategies[9-12]. Static load balancing uses pre-defined strategies. Task scheduling is independent of the current system load situation. Dynamic load balancing makes decisions on shifting tasks based on a system's real workload status. It can gain better efficiency, but it is more complex. Usually, single-point fail problem of scheduler node or processor node still exists. Task migration is a new trend in load balancing in order to provide certain fault tolerance. But single-point fail problem is not solved if the scheduler node crashes. The capability of providing fault tolerance is limited. The work presented in Ref. [13] adds a load balancing service to TAO, which conforms to CORBA specifications.

There is little literature on the combination of fault tolerance and load balancing due to the complexity of the problem itself. In Ref. [14], two front ends (FE) receive client requests, and these requests are transferred to the back ends of query ends (QE). Ref. [14] discussed five fault tolerant schemes between FE and QE and also strategies of load balancing.

The DCG in Southeast University implements a fault tolerant prototype with active replication[15,16]. Considering the features of load balancing, a novel framework of fault tolerance and load balancing is proposed. Further, three ways to select task scheduler group are given out. Our framework is basically different from those schemes in Ref. [14]. In our framework, fault tolerance is provided by active replication. All the requests are executed by a group. Tasks are scheduled by a group. A group is a basic processing unit. Regarding the schemes in Ref. [14], the main idea of fault tolerance is to migrate a server in case of failure. Our framework is more general. Five schemes are particular instantiations of our framework.

To our best knowledge, no literature has been published on this specific topic.

2 Problem

Active replication leads us to a way to enhance FT. The main idea of fault tolerance with active replication is to let more than one replica process one client request. Meanwhile, the principle of LB is to reduce the number of client requests on one server. These two techniques exert efforts on different dimensions. So, when a system encounters requirements of both FT and LB, the system needs to keep a balance between them. If we regard the efficiency function of FT and LB as A and B respectively, the problem is to find out the turning point of a joint function of A and B for its maximum value.

The difficulties involved in solving the problem are rooted in the differences between two techniques. Besides the contrast of the main technique idea mentioned before, the other key difference is that the global load information is not necessary for FT, but it is critical for LB. Therefore, any solution needs to answer at least two questions. The first one is what is a good redundancy for one server. Too many replicas in a server group may lead to heavier communication and other extra costs for each replica, while too few replicas in a group may violate FT. The second one is how to obtain global dynamic load information of server groups based on FT mechanisms.

This paper will illustrate our framework and show our answers to these two questions.

3 An LB Framework with FT

3.1 Assumptions

An asynchronous distributed system is composed of a set of pre-defined processors $\Omega = \{p_1, p_2, \ldots, p_n\}$ $(i = 1, 2, \ldots, n)$. All the processors are linked, and communications among processors depend on message transfer. There is neither upper bound for a message transfer, nor upper bound for a step execution on a processor. A processor is a correct processor if it works according to its specifications. Otherwise, a processor is a faulty processor. The system follows a fail/stop framework, which means any faulty processor will not participate in system computing any more when it crashes. A majority of processors in Ω are correct processors. Specifically, there are more than $\lfloor n/2 \rfloor$ correct processors.

Every processor p_i is equipped with a local failure detector D_i that satisfies $\diamond$S, which holds strong completeness and eventually weak accuracy properties.

We further assume that tasks $t_1, t_2, \ldots, t_m (m \geqslant 0)$ in an asynchronous distributed system are independent and non-preemptive. Every task is presented by a 5-tuple, i. e., $t_i = \langle \mathrm{cpu}_i, \mathrm{mem}_i, \mathrm{criticallevel}_i, \mathrm{starttime}_i, \mathrm{endtime}_i \rangle$, where the parameters stand for CPU occupation rate, predicated use of memory, task importance,

start time and end time from left to right respectively. A task scheduling module applies static LB strategies.

3.2 Framework architecture

The framework architecture is hierarchical. Three layers, namely network infrastructure (NI) layer, FT layer and LB layer are from bottom to top, as shown in Fig. 1. The NI layer is to provide communication services for the upper layers. It can be a LAN or Internet, for example. The FT layer includes group membership and consistency management modules.

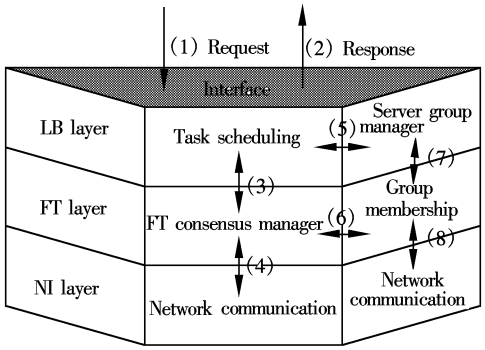

Fig. 1 Framework architecture

The LB layer covers server group management and task scheduling modules. We should point out that server group management involves manipulating the operations of joining, leaving, deletion and so on by the unit of a group rather than by a member. It is totally different from group membership in the FT layer. This paper pays more attention to the LB layer.

3.3 Server group management

All the groups supporting the FT and LB services form a server group. Server group management concerns the creation, split, mergence and deletion of FT groups based on the operation unit of an FT group. Operations of group split and group mergence depend on system loads.

① Group creation: Initially, a server group has one group member.

② Group split: When the server group manager decides to split group g_1 to form new $l(1 \leqslant l \leqslant n)$ groups $g_{11}, g_{12}, \ldots, g_{1l}$. The new group g_{11} will continue the work remaining by the former group g_1, and other new groups $g_{12}, \ldots, g_{1l}$ will get their transferred states and accept orders from the task scheduler.

③ Group mergence: When the server group manager decides to merge $k(k>1)$ groups, namely $g_1, g_2, \ldots, g_k$ with each group of m_y members, to be a new group g_1, the former g_1 is selected to be the base group, and other groups $g_2, \ldots, g_k$ join g_1. Before successful mergence, the groups $g_2, \ldots, g_k$ will not accept any new requests. After they finish their tasks and join a new group, the new group has $\sum_{y=1}^{k} m_y$ members. At the same time, the former g_1 will transfer states to all new members.

④ Group deletion: If all the members in a group crash or a group joins a new group, the group will be deleted.

3.4 Task scheduling module

The aim of task scheduling is to adjust task load dynamically in terms of FT groups. If system performance decreases, or if there are too many tasks in an FT group, or if response time is longer than threshold v, the task scheduler may make a decision to split groups. Meanwhile, if system load is rather light, or there are minimum replicas in an FT group corresponding to a critical service, the task scheduler may decide to merge groups. Redundancy of an FT group is related to the threshold values set in a specific system.

The task scheduler undertakes all the responsibilities of the task scheduling module. In a non-FT system, there is always only one task scheduler. This may lead to a single-point failure in a system. In our framework, there is a task scheduler group running in a system. From the point of view of client applications, the task scheduler group is transparent. There is no difference from single task scheduler. Then we face a new question: how to select or form a task scheduler group? We will answer this question in section 4.

3.5 A typical scenario

Referring to Fig. 1, a typical scenario for a task is as follows. A task t sends a request to a system by interface (1); the task scheduling module inquires the current server group status by interface (5); server group status is maintained by interface (7); the task scheduler chooses a group and makes a decision to transfer the request to the group by interface (3); members of a server group communicate with each other by interface (4) and (8); group membership information is transferred by interface (6); request results are returned by interface (2).

4 Task Scheduler Group Selections

The task scheduler group (TSG) is a special FT group. It is responsible for allocating client requests, coordinating redundancy of FT groups and adjusting load balance. In this section three ways are proposed for TSG selection. The difficulty in selecting a suitable TSG depends on the TSG being capable of gaining global group load information in order to make appropriate decisions for tasks. In an asynchronous system, it is rather difficult to achieve this.

4.1 Selection 1—Centralized control fashion

In centralized control fashion, one of the FT groups, namely g_1, is separated to be the TSG as shown in Fig. 2. Clients send their requests to the TSG. The TSG allocates requests to g_2, …, g_l respectively. After request execution, replies are delivered to the TSG. Clients receive replies by the TSG.

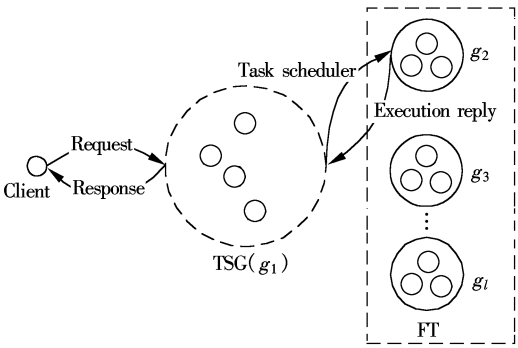

Fig. 2 Centralized control fashion

4.2 Selection 2—Round-robin fashion

FT groups act as the TSG in this fashion in turn. An FT group's turn is simply calculated by (#round mod #group). Each FT group has a continuous integer as its group number. If the result of (#round mod #group) equals a group number, the group becomes the current TSG. The initial value of #round is 0. It increases by 1 whenever the TSG is changed once. The value of #group equals members in the server group. If g_1 is the current TSG, clients send their requests to g_1. g_1 is in charge of allocation as shown in Fig. 3. If the TSG does not assign tasks to itself, we denote it as selection 2-1. Otherwise, it is selection 2-2.

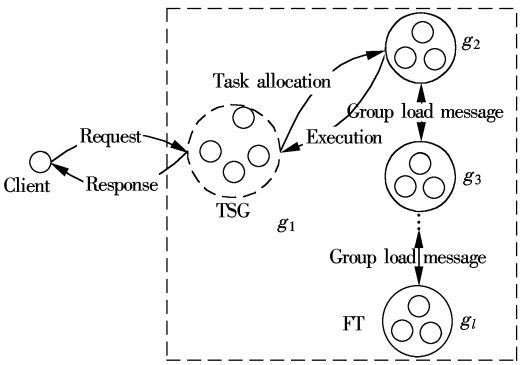

Fig. 3 Round-robin fashion

4.3 Selection 3—Inter-group fashion

The TSG in selection 3 is not a simple FT group. The TSG is composed of representatives of all FT groups in the server group. That is to say, a member of each group is selected as group representative to be a member of the TSG. If a member of the TSG fails, a new member, belonging to the same group as the crashed one, is selected and substitutes the former one. The work fashion is shown in Fig. 4.

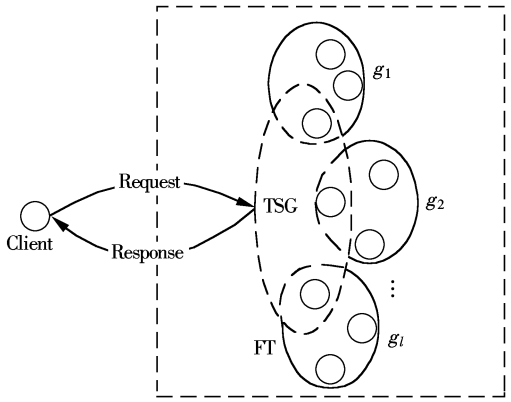

Fig. 4 Inter-group fashion

4.4 Comparisons of selections

In order to evaluate these selections, we first discuss some criteria.

1) Centralized control and decentralized control. Centralized control refers to a way that tasks are scheduled by one pre-defined and fixed processor; decentralized control is a way to schedule tasks by multiple processors.

2) Communication costs. This refers to how many messages are needed before a task is executed after it is received by the task service provider.

3) Total execution time for a task set.

4) Server fairness. Server fairness is good if the number of tasks executed by each server is almost the same. Otherwise, server fairness is bad.

5) Mean server throughput. This refers to mean tasks executed by a server in a unit time.

Secondly, we give some qualitative comparisons for these selections. Quantitative comparisons will be presented in section 5.

1) Selection 1 keeps the advantages of high throughput and simple control mechanism, and it avoids single-point failure. But the TSG may be a bottleneck in system performance. System load distribution is centrally controlled.

2) Selection 2 is good at multiplexing a group as the TSG. No more central control is needed. It is a decentralized structure, which is able to solve a performance bottleneck. Extra load messages need to exchange among groups as a cost. What's more, it is more complex to select the TSG in the case of group collisions.

3) Each member in TSG in the selection 3 has folded roles. It is a member of a group, and a member of the TSG. The task scheduler group is able to benefit from such folded role members. When the TSG receives client requests, it is not necessary to transfer re-

quests to certain group by the TSG, nor does the TSG deliver replies to clients with extra actions. A group's task load is brought to the TSG by the group's TSG member. It is able to achieve rather high performance. The drawback of the fashion is that the TSG management becomes complex.

5 Simulation Tests

In order to check the aforementioned analysis, simulation tests for selection methods are made.

5.1 Test environment

We assume that a system is composed of four FT groups. Each FT group has the same formation. Further, we assume that an FT service with active replication exists. Their effects will not be considered in the tests. Only costs due to load balancing are under consideration.

The TSG in selection 1 and selection 2-1 does not assign tasks for itself, while in selection 2-2, the TSG regards itself the same as other FT groups. For simplicity, we implement an FT group by a thread. Each task executes for 1 min. The strategy for task assignment is shortest-queue-first.

In simulation tests, all the FT groups run in one PC machine. The PC runs Windows XP. Its CPU is P4 1.8 G with a RAM of 256 MB. Programs are written with Visual C++6.0.

Communication costs are considered in selection 2. We assume that a network transmission delay is not longer than 5% of a task execution time, that is to say, 3 s. Network transmission delay is produced by a random function.

5.2 Test result and analysis

5.2.1 Performance of task arrival intensity

We observe mean server throughput in the conditions of determined single task execution time (1 min), task set size (including 6 000 tasks) and running period (250 min) and variable task arrival intensity (task/min). T1, T2, T3 and T4 in the following tables stand for thread 1, thread 2, thread 3 and thread 4, respectively.

Selection 1 Centralized control fashion

From Tab. 1, we observe that server fairness is rather poor in the case of lower task arrival intensity. This is because a server has completed its task before a new task arrives and hence the new task goes to the same server. It is not very difficult to improve this by accounting tasks assigned to a server. Server fairness is good with high task arrival intensity. We also notice that mean server throughput reaches its saturation state when task arrival intensity is about two times the number of servers in a system.

Tab. 1 Mean server throughput in selection 1

Tasks arrival per minute	Number of tasks executed			Throughput/ (task · min^{-1})
	T1	T2	T3	
2	249	189	1	1.756
3	249	243	130	2.488
4	249	248	247	2.976
5	249	249	248	2.984
6	249	248	248	2.984
7	249	248	248	2.980
8	249	247	248	2.976
9	249	248	249	2.984
10	249	248	248	2.980

Selection 2 Round-robin fashion

Selection 2-1 The TSG does not assign tasks to itself

From Tab. 2, we notice that in server fairness, selection 2-1 is worse than selection 1. In selection 2-1, there are four servers and mean server throughput is only near two. Two reasons lead to this phenomenon. One is that the TSG does not assign tasks for itself. This may waste some computing resources. The other is that the TSG needs to communicate with all the other servers in order to obtain current load distribution. A system has to pay extra costs for this.

Tab. 2 Mean server throughput in selection 2-1

Tasks arrival per minute	Number of tasks executed				Throughput/ (task · min^{-1})
	T1	T2	T3	T4	
2	189	91	71	52	1.612
3	181	137	105	98	2.084
4	182	147	103	98	2.120
5	194	147	89	104	2.136
6	190	140	100	91	2.084
7	188	145	103	90	2.104
8	180	148	102	100	2.120
9	180	140	100	95	2.060
10	175	145	99	97	2.064

Selection 2-2 TSG assigns task for itself

From Tab. 3, it is obvious that mean server throughput in selection 2-2 is better than that in selection 2-1 due to different strategies.

Tab. 3 Mean server throughput in selection 2-2

Tasks arrival per minute	Number of tasks executed				Throughput/ (task · min^{-1})
	T1	T2	T3	T4	
2	188	157	89	4	1.752
3	185	187	158	82	2.448
4	186	200	189	185	3.040
5	190	202	186	190	3.072
6	205	185	192	184	3.064
7	199	187	179	191	3.024
8	196	201	191	190	3.112
9	202	188	192	182	3.056
10	191	194	189	196	3.080

Selection 3 Inter-group fashion

From Tab. 4, we observe that mean server throughput is greatly improved in selection 3. It is nearly the extreme value of four with four servers in a

system. The explanation for this is that, on the one hand, all the servers make contributions to task execution. On the other hand, TSG members directly obtain the information of task load distribution for servers. No extra communication is needed.

Tab. 4 Mean server throughput in selection 3

Tasks arrival per minute	Number of tasks executed				Throughput/ (task · min^{-1})
	T1	T2	T3	T4	
2	249	185	0	0	1.752
3	249	241	128	1	2.476
4	248	248	211	63	3.080
5	248	248	244	178	3.676
6	249	248	248	246	3.964
7	249	247	248	248	3.968
8	248	248	246	246	3.952
9	249	248	247	248	3.968
10	249	248	248	248	3.972

We show the relationships of all selections on mean server throughput with Fig. 5. From Fig. 5, it can be seen that the common features for all the selections are as follows : ① Server fairness goes better when task arrival intensity goes higher; ② Mean server throughput reaches its peak when task arrival intensity is about two times the server numbers in a system. Selection 3 is the best among all the sections on mean server throughput.

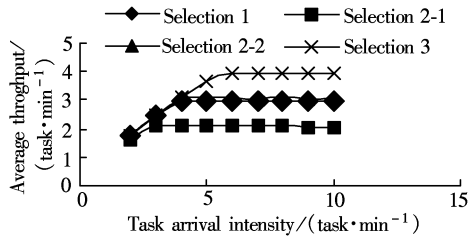

Fig. 5 Mean server throughput comparison for all selections

5.2.2 Performance of task set size

We observe the relations between total task execution time with task set size in the conditions of determined single task execution time (1 min) and task arrival intensity (6 task/min).

From Fig. 6, we notice that for all selections total task execution time goes up while task set size increases, especially at the linear rate. Among all the selections, selection 2-1 goes up at the fastest speed while selection 3 is the slowest one. Total task execution time in selection 3 is less than about 25% of that in selection 2-1. Total task execution time of selection 2-2 and selection 1 is similar.

It shows that we are able to calculate performance thresholds for a system if server capabilities are clear.

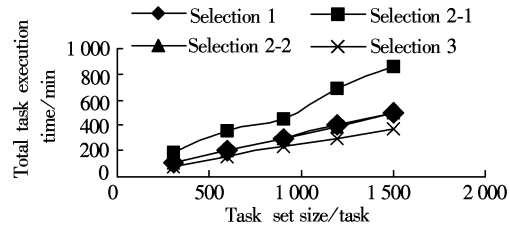

Fig. 6 Total task execution time with variable task set size for all the selections

5.2.3 Performance of single task execution time

We test and observe the relationships between total task execution time (min) and single task execution time in the conditions of determined task arrival intensity (6 task/min) and task set size (1 200 tasks).

Selection 1 Centralized control fashion

Tab. 5 shows that with the extension of single task execution time, server fairness is improved and total task execution time increases slowly. The costs due to task scheduling decreases from 29.22% (in the case of single task execution time of 10 s) to 5.16% (in the case of single task execution time of 80 s).

Tab. 5 Total task execution time with varied single task execution time in selection 1

Execution time/min		Number of tasks executed		
Single task	Total task	T1	T2	T3
10	282.583	1045	138	17
20	291.667	705	429	66
30	297.333	548	421	231
40	282.233	421	407	372
50	335.483	401	400	399
60	401.917	400	400	400
70	469.35	401	400	399
80	536.1	401	400	399

Selection 2 Round-robin fashion

Tab. 6 and Tab. 7 show that total task execution time does not go up rapidly until single task execution time reaches 40 s. This is consistent in the situation of four servers and task intensity of one task arrival every 10 s. It means that in this situation, a system goes to its saturation state, i. e., a server finishes a task execution as soon as a new task arrives. In the aspect of total task execution time, selection 2-1 is not so good as selection 1. Selection 2-2 is better than selection 2-1.

Selection 3 Inter-group fashion

Tab. 8 shows total task execution time with varied single task execution time in selection 3. Different selections are compared in the aspect of total task execution time with varied single task execution time in Fig. 7.

Tab. 6 Total task execution time with varied single task execution time in selection 2-1

Execution time/min		Number of tasks executed			
Single task	Total task	T1	T2	T3	T4
10	289	825	171	120	57
20	409.417	822	178	120	80
30	523.917	696	234	141	129
40	562.983	520	288	198	194
50	608.717	499	284	210	207
60	662.617	463	300	218	219
70	726.217	452	316	215	217
80	852.733	440	317	221	222

Tab. 7 Total task execution time with varied single task execution time in selection 2-2

Execution time/min		Number of tasks executed			
Single task	Total task	T1	T2	T3	T4
10	282.917	661	397	127	15
20	290.083	537	355	242	66
30	287.233	398	352	286	164
40	287.75	315	310	310	265
50	329.316	319	295	292	294
60	395.25	311	289	315	285
70	458.833	296	304	292	308
80	516.25	296	303	307	294

Tab. 8 Total task execution time with varied single task execution time in selection 3

Execution time/min		Number of tasks executed			
Single Task	Total task	T1	T2	T3	T4
10	282.083	939	252	9	0
20	280.4	615	410	174	1
30	282.917	526	408	257	9
40	281.4	419	394	333	54
50	292.5	340	336	320	204
60	302.433	301	301	300	298
70	352.333	301	301	300	298
80	402.583	301	300	300	299

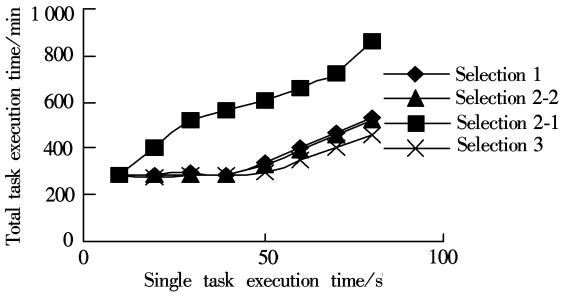

Fig. 7 Relations between total task execution time and single task execution time

Fig. 7 shows the trend that total task execution time goes up with the increase of single task execution time. All the curves go up at different speeds. A single task execution time of 40 s is a turning point in system performance in the specific system configuration. Of course, the specific value of the turning point is related to a specific system. In the aspect of total task execution time, selection 3 is about 80% of selection 2-2 and 47.2% of selection 2-1.

5.3 Analysis summary

① The performance of selection 3 is the best, then that of selection 1 followed by selection 2-2. Performance of selection 2-1 is the worst.

② The main reason for the poor performance of selection 2-1 is that extra communication is needed in order to obtain a current load distribution of fault tolerant groups. In fact, test data show that this extra cost is possible to compensate for. For example, if a new strategy is applied, such as selection 2-2, the performance is obviously improved and may go to the level of that of selection 1.

③ Mean server throughput in all the selections follows the trend of task intensity. It reaches a saturation state when task intensity reaches a threshold. Usually, the threshold is two times the number of servers in a system.

④ Total task execution time has a linear relation to single task execution time. Turning points exist in the system performance. These turning points are helpful for designing distributed systems.

6 Conclusion

This paper analyzes the pros and cons of both techniques of fault tolerance with active replication and load balancing. A novel framework of load balancing based on fault tolerance with active replication is presented. Three selection methods for task scheduler groups are addressed and compared. The framework is for independent and non-preemptive tasks and applies static load balance strategies.

In the future we will implement a prototype of the framework and quantificationally analyze group redundancy, and we will also extend the framework for preemptive tasks and dynamic load balance strategies.

References

[1] Polledna S. *Fault tolerant real-time systems: the problem of replica determinism* [M]. Boston: Kluwer Academic Publishers, 1995.

[2] Chandra J, Toueg S. Unreliable failure detectors for reliable distributed systems [J]. *Journal of the ACM*, 1996, **43**(2): 225 – 267.

[3] Fischer M J, Lynch N A, Paterson M S. Impossibility of distributed consensus with one faulty process [J]. *Journal of the ACM*, 1985, **32**(2): 374 – 382.

[4] Wensley J H, Lamport L, Goldberg J, et al. SIFT: design

and analysis of a fault-tolerant computer for aircraft control [J]. *Proceedings of the IEEE*, 1978, **66**(10): 1240 - 1255.

[5] Schneider F B. Implementing fault-tolerant services using the state machine approach: a tutorial [J]. *ACM Computing Surveys*, 1990, **22**(4): 299 - 319.

[6] Kenneth B, Thomas J, Frank S. ISIS: a distributed programming environment, version 2.1—user's guide and reference manual [EB/OL]. http://www.cs.cornell.edu/Info/Projects/ISIS/ISISpapers.html. 1987/2005-06-05.

[7] Renesse R, Birman K, Maffeis S. Horus: a flexible group communication system [J]. *Communications of the ACM*, 1996, **39**(4): 76 - 83.

[8] Peterson L L, Bucholz N C, Schlichting R D. Preserving and using context information in interprocess communication [J]. *ACM Transactions on Computer Systems*, 1989, **7**(3): 217 - 246.

[9] Kanmeda H, Li J, Kim C, et al. *Optimal load balancing in distributed computer systems* [M]. London: Springer-Verlag, 1997.

[10] Zhou S. A trace-driven simulation study of dynamic load balancing [J]. *IEEE Transactions on Software Engineering*, 1988, **14**(9): 1327 - 1341.

[11] Kostin A E, Aybay I, Oz G. A randomized contention-based load-balancing protocol for a distributed multiserver queuing system [J]. *IEEE Transactions on Parallel and Distributed Systems*, 2000, **11**(12): 1252 - 1273.

[12] Eager D, Lazowska E, Zahorjan J. The limited performance benefits of migrating active processes for load sharing [J]. *ACM SIGMETRICS Performance Evaluation Review*, 1988, **16**(1): 63 - 72.

[13] Othman O, O'Ryan C, Schmidt D. The design of an adaptive corba load balancing service [EB/OL]. http://www.cs.wustl.edu/~schmidt/PDF/load_balancing2.pdf. 2001/2005-06-05.

[14] Friedman R, Mosse D. Load balancing frameworks for high-throughput distributed fault-tolerant servers [J]. *Journal of Parallel and Distributed Computing*, 1999, **59**(3): 475 - 488.

[15] Wang Y. Active leave behavior of members in a fault-tolerant group [J]. *Science in China Ser F Information Sciences*, 2004, **47**(2): 260 - 272.

[16] Wang Y, Wang J L. A novel load balancing framework for active replicated servers in asynchronous distributed systems [A]. In: *The 16th IASTED International Conference on Parallel and Distributed Computing and Systems* [C]. Cambridge, 2004. 298 - 303.

一种面向主动复制服务器的负载平衡框架

汪 芸 王俊岭

(东南大学计算机科学与工程系,南京 210096)

摘要:研究解决了在分布式系统中同时提高系统可靠性和运行效率的问题.针对基于主动复制的容错技术和负载平衡技术,分析了这2种技术的优势和劣处,提出了一种基于主动复制容错的负载平衡框架,讨论了该框架的层次结构.该框架能够根据系统负载,动态地调整系统中容错组的个数以及容错组中成员的个数.提出了3种选择任务调度组的方法,并进行了仿真测试.通过对仿真测试数据的分析,对任务到达强度、任务集大小以及单个任务执行时间与任务集执行时间的关系进行了讨论,这些分析结论将有助于分布式系统的设计.

关键词:负载平衡;容错;框架;任务调度组

中图分类号:TP391