A Scalability Performance Study for General-Purpose Applications on HTCaaS :The Database Perspective

Seungwoo Rho*, Jik-Soo Kim**, Sangwan Kim***, Seoyoung Kim****, Soonwook Hwang*****
KISTI, Korea
E-mail : seungwoo0926@kisti.re.kr*, jiksoo.kim@kisti.re.kr**, Sangwan@kisti.re.kr***
sssyyy77@kisti.re.kr****, hwang@kisti.re.kr*****

1. Introduction

HTCaaS(High-Throughput Computing as a Service)[1] is a pilot-job[2] based multi-level scheduling system that enables scientists to solve their complex and demanding scientific problems. We are currently exploiting a centralized relational database to record and monitor the status of application tasks and pilots running. We also support web services that can be utilized for communication between different components in HTCaaS. However, as we increase the number of computing resources concurrently connected to our HTCaaS server, the number of threads processing database queries and the communication traffic between database and server become substantial. This means that the scalability of our HTCaaS server can depend on the processing capability of the database machine so that the clock speed and memory size of the database machine can limit the maximum throughput of our system. In this paper, we present detailed analysis of performance impacts of our database system in HTCaaS and show directions of improving scalability of our system to support more challenging scientific applications.

2. Scalability Experiments of HTCaaS on PLSI

In this section, we briefly introduce our HTCaaS system and present our work on optimizing and tuning database configurations and experimental results that can address scalability issue to accommodate thousands of computing cores in supercomputing infrastructures in Korea.

HTCaaS consists of 7 different modules which can be categorized into agent management and service management (providing web service APIs). The agent management is responsible for job scheduling & processing, resource assignment and agent deployment, and consists of Agent Manager and agent (pilot-job). The service management provides web services to communicate with each other components and includes Account Manager, Database Manager (based on MySQL), Monitoring Manager, Userdata Manager and Job Manager. HTCaaS adopted OGF JSDL[3] standard as a job description language which provides parameter sweep functions to enable scientists to submit a large of jobs conveniently and also provides an easy-to-use GUI/CLI/Web portal. HTCaaS is currently running as a *pilot service* on top of PLSI (Partnership & Leadership for the nationwide Supercomputing Infrastructure)[4] which provides researchers with an integrated view of geographically distributed supercomputing infrastructures in Korea. The total computing resources of PLSI are about 7,448 cores, linked with General Parallel File System (GPFS)[5] as a global shared storage and PLSI uses IBM LoadLeveler(LL) as a global scheduler.

Database optimization & tuning for the better performance typically consists of two kinds of methods, global and thread variable. Generally, global variable tuning method is carried out first and then thread variable tuning is performed. There are many metrics to tune database and optimal values can be found throughout long-term and regular database monitoring. In this paper, we only selected core metrics that can affect server performance the most significantly and set those values considering the amount of physical memory.

Table 1. The global variable

Global variable	default(byte)	change(byte)	usage
innodb_buffer_pool_size	134,217,728	8G	for only innodb
innodb_additional_mem_pool_size	8,388,608	8,388,608	for only innodb
innodb_log_buffer_size	8,388,608	8,388,608	for only innodb
innodb_log_file_size	5,242,880	1G	for only innodb
innodb_flush_log_at_trx	0	0	for only innodb
max_connections	151	500	both
key_buffer_size	8,388,608	1G	for only myisam
table_open_cache	64	1000	both
thread_cache_size	0	500	both
query_cache_size	0	1G	both

Table 2. The thread variable

Thread variable	default(byte)	change(byte)	usage
sort_buffer_size	2,097,144	2,097,144	both
read_buffer_size	131,072	131,072	both
bin_log_cache_size	32K	32K	both
thread_stack	256K	256K	both
join_buffer_size	131072	131072	both
tmp_table_size	16M	16M	both
myisam_sort_buffer_size	8,388,608	8,388,608	for only myisam

Among the global variable, max-connection is the most important factor for HTCaaS scalability, which is determined by the size of physical memory (our database machine has a 16GB of memory). The general formula[6] to compute this metric is shown in Table 3. The table 3 is for both MyISAM and InnoDB storage engines, but our system mainly uses InnoDB storage engine because this engine is more appropriate for many task processing. MySQL can also exploit until maximum 75%(12GB in our case) of all the physical memory if we exclude data cache/buffer.

The aim of this work is to show theoretical result for computing overall database access time in an agent execution and experiment results to clarify the correlation between agent and database connection. In this experiment, we do not consider job dispatch and data I/O time. Table 4 indicates average database access time of each method

(running and monitoring) used when one agent communicates with the database manager. Agent runs two kinds of threads (monitoring and running) and the monitoring thread calls each method periodically to check heart beat and exit signal. If we assume that the limit of database connection is 1 and two threads of an agent access database at the same time, the maximum access delay time is 0.429s (First sendAgentSignal+setAgentHost). So, if ten agents try to connect the same database with one limited thread, the maximum access delay time will be about 4.29s.

Table 3. The computation formula for core global variables

Physical_Memory > innodb_buffer_pool_size + key_buffer_size + $innodb_additional_mem_pool_size + innodb_log_buffer_size + $query_cache_size + max_connections*(sort_buffer_size + read_buffer_size + binlog_cache_size + thread_stack + $join_buffer_size + $tmp_table_size + $myisam_sort_buffer_size)
innodb_buffer_pool_size + key_buffer_size < 80% of physical memory

Table 4. Average access time of database-related methods in agent

	Method	Average time(ms)
Running Thread	getJobStatus	15.3
	setJobLog	53.3
	setAgentCurrentJob	22
	startJob	15.6
	finishJob	19.3
	setAgentHost	212.6
	startAgent	16
	finishAgent	14.3
Monitoring Thread(Periodic)	First sendAgentSignal	217.3
	Next sendAgentSignal	21
	checkAgentQuit	20
	setCEAliveAgentAdd	52.3

Figure 1shows the number of average database connections used when the number of agents increases from 25 to 400 on PLSI. Three color lines represent the length of a job execution (from 0, 30 to 60 seconds). Most of real applications we use have a job length more than 60 seconds and one of them has the job length of about 20 seconds. So, we selected three kinds of job execution times which can show that HTCaaS can support more challenging scientific applications. As the length of a job increases, the average growth of the database connection decreases. As we can see from gradients of green and red lines (60 and 30 seconds of job execution times respectively) in Figure 1, we can expect the quantity of stably manageable agents to be much larger since 400 agents consume only 27 connections (our database can deal with 500 connection threads). Also, as we increased the number of agents twice at each step of our experiments, the number of required database connections was quite sloping gradually, which implies that our system can deal with larger-scale agents (probably more than 7,448 which is the number of total CPU cores in PLSI). In addition, there is another solution to enhance agent scalability. It is to increase the monitoring interval of agents for heartbeat. Actually, the monitoring thread only accesses the database when the agent is processing a job and we set the heartbeat time 1 minute. Throughout our experiments, we showed that our HTCaaS system can be successfully developed into a production-level service on top of PLSI supercomputing infrastructures by effectively integrating thousands of CPU cores.

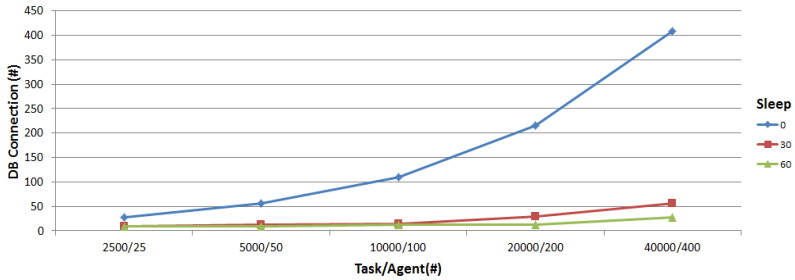

Figure 1. Average DB Connection

3. Conclusions and Future work

In this paper, we briefly introduced our HTCaaS system that can support large-scale scientific applications by leveraging distributed supercomputing infrastructures in Korea and presented detailed analysis of performance impacts of our database system. Future work is to reenact and prove this experiment with larger scale agents (more than 7,000) on a hybrid resource environment like cloud, grid and supercomputers.

4. References

[1] JS Kim, S Rho, S Kim, S Kim, and S Hwang, “HTCaaS: Leveraging Distributed Supercomputing Infrastructures for Large-Scale Scientific Computing”, 6th Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS) 2013, ACM, Denver, Colorado USA, 17-22 Nov. 2013
[2] Luckow, A., Santcroos, M., Mezky, A., Weidner, O., Mantha, P. and Jha, S., “P*: A model of pilot-abstractions”, 2012 IEEE 8th International Conference, IEEE, Chicago, 8-12 Oct. 2012, pp. 1-10
[3] Anjomshoaa, A., Brisard, F., Drescher,M., Fellows,D., Ly,A., McGough, S., Pulsipher, D., Savva, A., “Job Submission Description Language (JSDL) Specification,Version 1.0”, GFD-R.136. Open Grid Forum (2008)
[4] http://www.plsi.or.kr/
[5] http://en.wikipedia.org/wiki/IBM_General_Parallel_File_System
[6] http://dev.mysql.com/doc/refman/5.5/en/innodb-configuration.html