Evaluation of Software and Hardware Resource Topology of System Based on Request Processing Simulation

Bo YANG, Jun WEI, Qing-quan DONG and Cheng-gui CHU*

NARI Group Corporation, No. 19, Chengxin Avenue, Jiangning District, Nanjing City, Jiangsu Province, China

*Corresponding author

Keywords: Simulation, Performance evaluation, Data center, Request.

Abstract. In order to swiftly evaluate the influence of the modifications derived from system's software and hardware topology on the system performance, we proposed a method for system performance evaluation in this paper. Firstly, we divided the request-handling process of system into two phases: calling software components by requests and occupying hardware resources by component operation, then we constructed a probabilistic model for system behavior in these two stages by taking system topology and request distribution as variates, and at last we created a simulation algorithm based on this probabilistic model which could be employed in system performance evaluation. This evaluation algorithm can be used in scenarios where the hardware resources of system are sufficient and concurrent requests are in big number, at the same time, it can reduce the cost for and speed up the performance evaluation because it does not need to build a real system.

Introduction

Traditionally, when optimizing the structure of a data center system, it is usually considered to optimize the underlying hardware topology of the system to obtain the optimal or near optimal performance [1,2,3]. However, this optimization method can’t be used when changes happen in the software topology of the system or the distribution of requests received by the system. The computer system that can handle requests smoothly is composed of the lower hardware system and the software system which is compatible with that hardware system. The efficiency of handling requests in a computer system is not only affected by the performance of the hardware used by the underlying hardware system and the topological structure formed by the interconnection of the hardware, but also by the internal design of the upper software system and its deployment on the hardware system. Therefore, in order to evaluate the efficiency of a computer system in processing requests, we need to consider the above four aspects.

Therefore, in this paper we assume that we have the current system hardware resource topology, software resource topology, software resource deployment on hardware resources (these three aspects constitute the current system hardware and software topology), and the current request distribution information. The impact of these factors on system performance will be measured by throughput-related indicators [4].
System Analysis

The Request Distribution

The efficiency evaluation of the system is based on its performance indicators in processing requests. Thus, the first thing we need to simulate is request generation. If we regard the process of generating requests as a step by step one, then what kinds of requests are generated at each step and how many requests for each kind is the information we want to know, or for short, we need the distribution of requests to generate requests, in a simulation way.

In reality, the random generation of multiple types of requests is mostly independent of each other, and the current amount of generated requests has nothing to do with the history of past generation amounts. Therefore, it can be assumed that the number of generated requests $n_{req}$ at each step obeys the same normal distribution:

$$n_{req} \sim N(u_{req}, \sigma_{req}).$$

(1)

where $u_{req}$ is the mean value of generated amount in each step, and $\sigma_{req}$ is the variance of generated amount. We can control the generation of such requests by adjusting these two parameters. Multiple kinds of requests correspond to multiple normal distributions. As long as we have these probability distributions, we could construct the sequence of requests that the system needs to process in one simulation, so that the efficiency of system can be measured.

The efficiency of request processing of system is usually measured by three indicators: transaction per second, average response time and concurrency. The relationship between them is:

$$\text{transaction per second} = \frac{\text{concurrency}}{\text{average response time}}.$$  \hspace{1cm} (2)

Evaluating the efficiency of system here is achieved by measuring the above three indicators that the system shows during the request sequence handling process.

The Topology of System Hardware Resources

In order to reduce the hardware details needed to be taken into account, for hardware resources, only memory resources, external resources, computing resources and bandwidth resources are considered, and their respective performance is expressed by a number. System hardware resource topology is an undirected graph with four kinds of resources as vertices and data transmission relations as edges, and each vertex in the graph has an attribute value which indicates the amount of resources it possesses. Among these vertices, only computing vertices can exchange data with memory vertices directly, while other types of vertices (including another memory vertex) must exchange data with memory through bandwidth vertices as intermediaries.

Software Resource Graph of System

The software resources used in the system are represented by a software resource graph with hardware resource occupancy information in this paper. It consists of two parts: one is the software resource topology graph which is used for representing relationship between different software resources; the other one is the hardware resource occupancy information of software resources, which is used to express the relationship between software resources and hardware resources.
Software can be seen as a composition of components of various functions, data sets used by components and data connections between them. With components and data sets as vertices and data connection as edges, the system software resource topology graph is constructed.

Hardware resource occupancy information of software resources can also be divided into two parts. Since each software resource has two states: running and stationary, we need to describe their occupancy of hardware resources in both states. When software resources are static, components occupy a fixed amount of memory and external memory, while data sets occupy only a fixed amount of external memory.

When the software resources are in running state, it will not only occupy the fixed amount of memory and external resources because of deployment, but also need to increase the occupancy of various hardware resources dynamically since processing requests need more resources. In this paper, we ignore the dynamic process of frequent occupancy and release of resources during their operation, and simplify their occupancy of a resource when they are called into two random variables: the number of occupied resources and the length of occupied time. These random variables can be considered to obey normal distribution. Denote the amount of computational resources occupied as $r_C$ and the length of occupancy time as $t_C$; denote the amount of memory resources occupied as $r_M$ and the length of occupancy time as $t_M$; denote the amount of external resources occupied as $r_E$ and the length of occupancy time as $t_E$; denote the amount of bandwidth resources occupied as $r_N$ and the length of occupancy time as $t_N$. These random variables obey the following two-dimensional normal distributions, respectively:

$$\begin{align*}
(r_C, t_C) & \sim N(u_C, \sigma_C, \rho_C) \\
(r_M, t_M) & \sim N(u_M, \sigma_M, \rho_M) \\
(r_E, t_E) & \sim N(u_E, \sigma_E, \rho_E) \\
(r_N, t_N) & \sim N(u_N, \sigma_N, \rho_N)
\end{align*}$$

(3)

where in these formulations, $u_k, \sigma_k, \rho_k$ for $k \in \{C, M, E, N\}$ are the parameters for corresponding two-dimensional normal distributions. When components, data sets and data connections are used in a request handling process, the amount and time of hardware resources they need to occupy are sampled according to these probability distributions.

**The Deployment of Software Resources**

Software resources need to be deployed to hardware resources to form a system that can handle requests. Deployment is to specify the computing, memory and memory vertices in the hardware resource graph corresponding to each component, the memory vertices corresponding to each data set, and the bandwidth vertices corresponding to each data connection. With the deployment of software resources, you know which hardware resources to occupy when you simulate handling requests.

**Resource Subgraph of the Request**

When processing a request, the system needs components to perform operations, needs data sets to provide data and data connections for data communication. These combinations of components, data sets and data connections used in request handling are actually a subgraph of the software resource topology graph, which should be named as “resource subgraph corresponding to request” or “resource subgraph” for short.

When system processes a class of requests, its components, data sets and data connections in the
resource subgraph which corresponds to these requests are invoked with certain probability. Therefore, each vertex and edge in our resource subgraph has a value to represent the probability that they will be called when processing a request. When the simulation system is going to handling requests, it needs to generate those software resource for the request handling at random according to probabilities in the resource subgraph.

The Impact of Software Resource Execution Order on Simulation

At the time of processing a class of requests, system’s software resources in the corresponding resource subgraph are executed in some complex partial order [5]. If we take this complex partial order into consideration when doing the simulation, the complexity of problem will be prohibitive. In fact, if the total hardware resources of the system are enough, we can evaluate some indicators of the system by simulation without regard to the troublesome execution order.

**Proposition 1**: In the processing of randomly generated requests, the effect of the execution order of the corresponding software resources of requests on the concurrency of the system decreases with the increase of total hardware resources of the system.

This proposition shows that as long as the capacity of hardware resources of system is large enough, it is not necessary to consider the execution order of software resources when evaluating the concurrency indicator of the system.

What about another indicator, the avarage response time? When system handles a request, it usually has a component as the entry component for this request: it accepts the request when the request occurs, and then returns the result of the handling at the time of completion of request processing. Thus the time consumed by the entry component in processing the request is right the response time of this request. We have another proposition for this indicator.

**Proposition 2**: In the processing of randomly generated requests, as long as the capacity of hardware resource of system is large enough and the corresponding software resource subgraphs of the requests all have fixed entry components, then the execution order of the software resources has little effect on the average response time of the system.

That’s to say, when the total amount of hardware resources of the system is big enough and the requests waiting for handling are randomly generated, it makes no sense to consider the complexing running order of components, data sets and data connections when evaluating the two indicators of system. This result is applied in our algorithm and significantly simplifies the simulation of request handling process.

Evaluation Algorithm

**Input**: Hardware resource topology \( H \), software resource graph \( S \), the deployment of software resource graph \( D \), request distribution \( \{N(u_{reqk}, \sigma_{reqk})\} \), resource subgraphs \( reqR \), loop steps \( N \), time step size \( s \).

**Auxiliary variables**: request processing list \( processingList \), request waiting quest \( waitingQueue \), time \( t = 0 \), number of processed requests \( processedNumber = 0 \), average response time \( averageResponse = 0 \), amounts of different kinds of requests need to be generated \( requestNumbers \), generated request list \( requestList \).

Start.
loop from 1 to $N$:
   for $req$ in $processingList$:
      for $e$ in $req.runningMembers$
         release the hardware resources occupied by $e$ if expire time smaller than $t$
      if all $e$ in $req.softwareSources$ has released their hardware resources
         delete $req$ from $processingList$.
         $processedNumber = processedNumber + 1$
      
      $averageResponse = averageResponse \times \frac{processedNumber - 1}{processedNumber} + \frac{(t - req.startTime)}{processedNumber}$

for $k$ in $\{1, ..., n\}$
   $requestNumbers[k] = sample\left(N(u_{req_k}, \sigma_{req_k})\right)$

$requestList = generateRequest(requestNumbers, reqR)$
for $req$ in $waitingQueue$:
   $req = allocateHardwareResources(req, H, S, D)$.
for $req$ in $requestList$:
   $req = allocateHardwareResources(req, H, S, D)$.
   if all $e$ in $req.softwareSources$ have received hardware resource they need:
      put $req$ into $processingList$
   else:
      put $req$ into $waitingQueue$
   $t = t + s$
end loop

$transactionPerSecond = \frac{processedNumber}{t - s}$
$councurrency = transactionPerSecond \times avarageResponse$
return $transactionPerSecond$, $councurrency$, $avarageResponse$
End.

Function explanation:

$sample(reqD[k])$: Sample a number according to distribution $N(u_{req_k}, \sigma_{req_k})$.
$generateRequest(requestNumbers, reqR)$: Generate requests according to the number and type of request, where each request generated consists of software resources which are needed for handling this request.
$allocateHardwareResources(Req, H, S, D)$: Allocate hardware resources to software resources of request. This needs hardware information from $H$, software information from $S$, and the deployment information from $D$.

Conclusion
The system efficiency evaluation method described in this paper takes into account the impact of four factors, which are the request distribution, hardware resource topology, software resource topology and the deployment of software resources on hardware resources, on the request-processing efficiency of system. The above four factors are appropriately represented and modeled by several
topology graphs combing with probability method, and on the basis of this modeling, a simulation algorithm is constructed to simulate the running process of the system to evaluate the TPS, response time and concurrency of the system. This method can effectively estimate the alteration of the system request-processing efficiency caused by the changes of above four factors, and should be used in the scenario where the system resource capacity is sufficient so that the amount of concurrent request is big enough to almost erase the influence of executing order of components and data sets on hardware resources allocation.

Acknowledgment
This research was financially supported by the Science and Technology projects of State Grid Corporation of China (No. 500623723).

References
[1] Nakao M, Murai H, Sato M. A Method for Order/Degree Problem Based on Graph Symmetry and Simulated Annealing with MPI/OpenMP Parallelization[C]//Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region. ACM, 2019: 128-137.

[2] Valadarsky A, Shahaf G, Dinitz M, et al. Xpander: Towards optimal-performance datacenters[C]//Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies. ACM, 2016: 205-219.

[3] Shafigh A S, Lorenzo B, Glisic S, et al. A framework for dynamic network architecture and topology optimization[J]. IEEE/ACM Transactions on Networking, 2015, 24(2): 717-730.

[4] Jyothi S A, Singla A, Godfrey P, et al. Measuring and understanding throughput of network topologies[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press, 2016: 65.

[5] Lamport L. Time, clocks, and the ordering of events in a distributed system[J]. Communications of the ACM, 1978, 21(7): 558-565.