International Journal of Database Theory and Application
Vol.9, No.3 (2016), pp.123-136
http://dx.doi.org/10.14257/ijdta.2016.9.3.14

# A System Performance Estimation Model for Cassandra Database

Bo-Qian Wang[1], Qi Yu[1], Xin Liu[1], Li Shen[1] and Zhi-ying Wang[1],

[1] *National University of Defense Technology, Changsha, Hunan*
*wangboqian_vip@163.com*

***Abstract***

*Cassandra database system is one of the universal databases. To achieve high performance, we should allocate memory space rationally according to actual demands. Otherwise, it will influence reading and writing performance. Actually, we always allocate memory space according to experience and repeated attempts which usually won't give us the best answer. To solve this problem, firstly we analyze the reading and writing processing of the Cassandra database and find out the corresponding memory space which will influence system performance. Secondly, we build up a relationship model between system performance and memory allocation and name it as The Memory Model of Reading and Writing Performance. We have already applied the relationship model to real database servers to guide memory allocation and performance evaluation. Simulation results show that this memory model could well describe the quantization relationship of memory space and system performance.*

***Keywords****: Cassandra database, Performance, Model, Quantization*

## 1. Introduction

Cassandra database system is a typically distribute NoSQL database system which developed by the Facebook. It is based on Dynamo and column storage method of Google BigTable, so that we can also call it Dynamo2.0. Nowadays it is one of the top projects of Apache and listed in the Top10 of the most popular database system. Because of the fascinating scalability, it is accepted by many famous Web2.0 websites such as Digg and Twitter [5-6].

Cassandra database mainly use the key-value model for the data storage, the Memtable data structure for the memory storage and the SStable data structure for the disk storage [7]. In this article, we firstly find out the main factors and their corresponding parameters configuration which directly affect the reading and writing performance of the database system. Secondly, we initially establish a quantitative model of memory allocation and reading and writing performance respectively. Finally we improve this model and build up a more practically one according to the hardware condition and user demands. Test results show that this model can accurately reflect the relationship between memory configuration and system performance and help us to allocate the memory space more reasonably.

## 2. Process Analysis of Cassandra Database

This part will describe the reading and writing process of Cassandra database system in details. By this way, we can find out the factors and configuration parameters related to the reading and writing performance in memory space.

ISSN: 2005-4270 IJDTA

### 2.1. Analysis of Data Process

Cassandra database response to the users' data requests by listening port 2160. After receiving users' data requests, the database system will create a new thread from the thread pool to deal with this request. Then Cassandra will make sure of the server which the data should be stored in and sent the data request to this server by port 7000. Finally the reading and writing process will be executed in the server. This article will mainly discuss the running mechanism in a single server. In the end we will build up a memory allocation model to solve the problem of memory allocation in the practical application process. The finally test indicates that this model can improve the whole performance of servers.

### 2.2. Analysis of Writing Process

The response to the writing request of Cassandra database is shown in Figure 2.

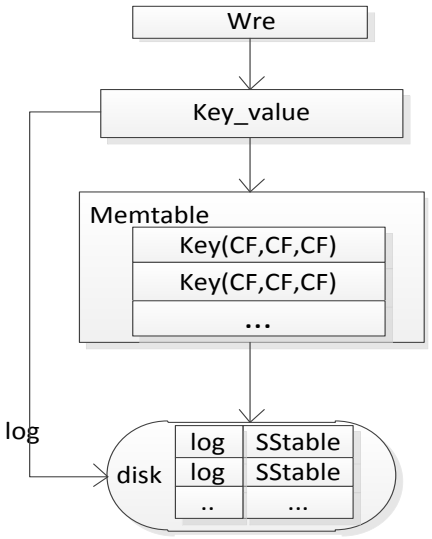



**Figure 2. The Process of Writing Request**

In the writing process, Firstly user's operation will be recorded in the log file to make sure that all the system record can be restored after unknown power off. Secondly, all the data written in will be recorded as key-value structure and then writing in the Memtable data structure in the JVM heap. The data that in the same column family will be stored in the same Memtable data structure. After the Memtable reaches a certain limit which we can set by modifying the configuration parameter, the management process will flush all the data in this Memtable data structure into disk and stored it as SStable data structure. Similarly, modifying requests and deleting requests is the same as the writing requests. By adding the record that indicates a certain data is modified or removed in the Memtable data structure, the real data will be changed or deleted when the SStable data structures finally combined. The mechanism discussed above decreases the disk operation of writing, modifying and deleting requests. So it promote the response throughput of user's requests.

We can come to the conclusion that the memory functional unit related to writing request is JVM heap and the Memtable data structure in it.

### 2.3. Analysis of Reading Process

Compared to writing process, reading process is more complex. A record may be stored respectively in Memtable or in different SStables which may keep the old or the

latest record. So as to a certain reading request, Cassandra database system must search all the record whatever it is the old one or the latest one. All the records found before will be emerged together to get the latest one according to the time label. The latest record will send back to user in the final step. The process is shown in Figure 3.

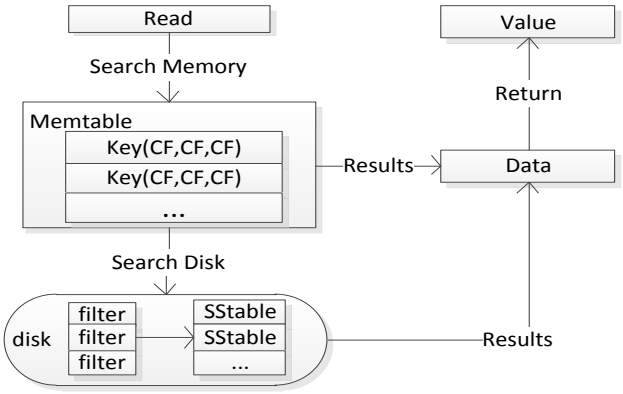



**Figure 3. The Process of Writing Request**

In the reading process, Cassandra database system will search results in the Memtable data structure according to the row_key value. After then, system will also search another results in the SStable data structures. Finally, all the results will be combined in the memory space and the latest record will be sent to the user. It has a lot of disk operation during this process. The first disk operation finds out the certain SStable data structures which include the key value user requests for. It has to search bloom filter file stored in disk. The second one will read the SStable into memory according to the result gotten in the first step [8-9]. To decrease disk operation, Cassandra database system adds row_cache and key_cache mechanism of which the cache file will be stored in memory space. The key_cache is a mechanism that stores the key value into memory space. So that Cassandra database system will search the key_cache to locate the data user requests in the first step. And if it hit in the key_cache, then the first disk operation will be avoided. As a result, the reading process will be sped up. What is more, because that the key value is very small, the key_cache mechanism will not take too much memory space. Similarly, the row_cache is also a mechanism that store the final data which is often requested into to the memory space before return it to user. If a user request hit the row_cache, the system will immediately return the result without the two disk operations. This mechanism will decrease the reading request to a large extent. However, because the row_cache has to store the final results which is much larger than the key value, it will take too much memory space.

Our test operation system is Ubuntu14.04. It has a memory optimization mechanism named cached memory which will cache all the files that have been read from disk into memory space or will be written into disk. By this mechanism, memory space can be made best use of and disk access will also be decreased. Finally it will help to increase the response throughput.

We can come to the conclusion that the memory functional unit related to reading request is key_cache, row_cache and cached memory.

Therefore, the memory space can be divided into three part: JVM heap, Cache and the space taken by the other system. JVM heap includes Memtable data structure and the cache can be further divided into key_cache, row_cache and cached memory. The space that taken by the other system is only related to the services provided by the operation system. So we should set it as a constant in a Cassandra server. The detail information is shown in Figure 4.

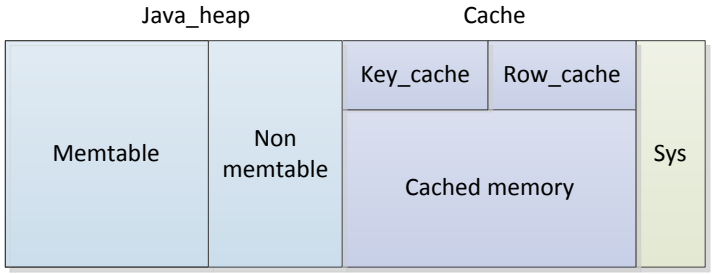



**Figure 4. Memory Allocation of Cassandra Database Server**

## 3. Method and Model

In this part will firstly discuss about the analysis of special case and make preliminary conclusion about the relationship between different memory functional unit and system performance. Secondly, we will respectively build up the quantitative relation model for the reading and writing performance in the infinite memory space circumstance. In the end, we will put forward a more practical and general model to guide the practical application. The hardware and software circumstance is listed in Table 1.

**Table 1. Hardware and Software Circumstance**

| | |
|---|---|
| | **HARDWARE CIRCUMSTANCE:** |
| 1 | CPU Intel(R) Core(TM) i7-2600 3.40GHz, four cores |
| 2 | L1 Cache 256KB*4,L2 Cache 1MB*4, LLC Cache 8MB*1 |
| 3 | Memory DDR3 1600MHz |
| | **SOFTWARE CIRCUMSTANCE:** |
| 1 | Ubuntu Desktop 14.04, kernel version 3.13.0 |
| 2 | Java version 1.8.0_20 |
| 3 | Cassandra 2.1.0, YCSB 0.1.4, data access：zipfian |

### 3.1. Preliminary Test and Analysis

At the beginning of the test, we firstly focus on the specific case to find out the basic relationship between memory allocation and system performance. The test condition is 8GB memory, single disk and single column family. All the other parameters will keep the default settings. The test result is shown as Figure 4.

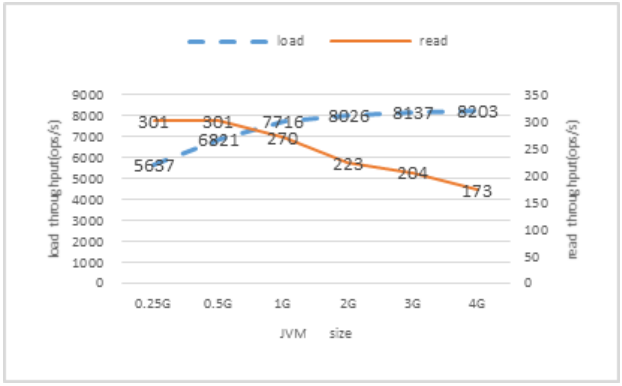



**Figure 5. Relationship between the JVM Heap and System Performance**

From Figure 5 we can draw the conclusion that the size of JVM heap has a great deal of influence on the reading and writing throughput. With the size of the JVM heap being larger, the size of cache space will become smaller on the contrary. This makes a lower reading throughput but a higher writing throughput. In this Figure we can clearly see that the 1GB JVM heap size will produce the best system performance. If we set it larger, it will make less contribution to the writing throughput but will speed up the decreasing process of reading throughput. In contrast, it will make less contribution to the reading throughput but will speed up the decreasing process of writing throughput.

Based on the test and analysis above, we can come to the basic conclusion that:

1. The writing throughput will increase with the size of JVM heap being larger. But this tendency will be less obvious.

2. The reading throughput will increase with tie size of cache space being larger. But this tendency will also be less obvious.

This simple test shows that different memory allocation strategy will influence the system performance. So it is more important to formulate the memory allocation strategy according to the certain request. Then, we will respectively build up the quantitative relation model for the reading and writing performance in the infinite memory space circumstance.

### 3.2. Reading and Writing Model In Infinite Memory Space

To find out the influence of cache space and JVM heap on the system performance, we assume that the memory space of server is infinite. Under this hypothesis, we can consider their influence separately and avoid their restricting relationship caused by the limited memory space. We will final build up models for the reading request and writing request respectively.

#### 3.2.1 The Influence of Cache on the Reading Performance

In this part, we will further discuss the quantitative relationship between the size of cache and reading throughput and build up related model.

Firstly, we try to find out the influence of key_cache and row_cache on the reading performance. The test environment is 10GB data without redundancy. And we can calculate the size of key_cache is about 880MB. Making sure that the size of the total cache space is unchanged, we gradually increase the key_cache space. The result is shown in Figure 6.

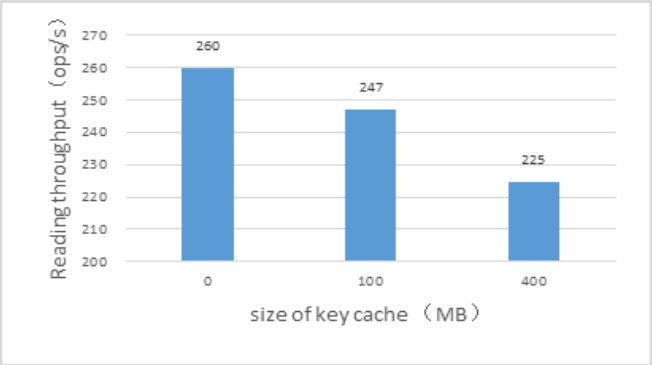



**Figure 6. The Influence of Key_Cache Size on the Reading Performance**

As shown in Figure 6, the increase of key_cache size will finally decrease the reading throughput under the circumstance that the total space of cache is unchanged. The key_cache mechanism provided by the Cassandra database will decrease the first disk access which will locate the data user request. But at the same time, the Linux has the

similar function called cached memory. This conflict will result in data redundancy and will also waste a lot of cache space which will decreases the valid data, increase the memory miss rate and the amount of disk access and finally decrease the reading throughput. So the key_cache mechanism should be avoided and set the corresponding parameter as 0MB.

Similarly, row_cache mechanism caches the finally results which often accessed by users. Although this mechanism can avoid two disk access, it is also the same as cached memory. Compared to the key_cache mechanism, it results in much more data redundancy which will seriously decrease the reading throughput. So the row_cache mechanism should also be avoided.

Therefore, only cached memory should be included in the cache space. So we will test on the cached memory below. The total amount of data is 10GB, the amount of reading is $10^5$. The test result is listed in the Table 2.

**Table 2. The Relationship between Cached Memory Space and Reading Performance**

| cached_mem（GB） | Running time（s） | Discrepancy of time（s） | performance promotion（s/GB） |
|---|---|---|---|
| 6.5 | 332 | | |
| 5.8 | 371 | 39 | 56 |
| 4.8 | 448 | 77 | 77 |
| 4.1 | 491 | 43 | 61 |
| 3.1 | 577 | 86 | 86 |

Theoretically, the total running time includes two parts. One is the time that spent on the memory data access and the other is on the disk data access. For a single data operation, the disk process is much longer than the memory process. But the data access matches the rule of Zipfian which is expressed as $fR = C$ ( $f$ is the frequency of data access, R is the rank of data and C is a constant). Referring to this formula, the majority of the data access operations happen in the memory. That is, in another word, the amount of operations happened in the memory is much more than those in the disk. So compared to the disk access time, the memory access time can't be ignored, either. So the total time for a reading request can be expressed as formula 1.

$$time = ops * memTime * f_1 + ops * diskTime * f_2 \quad (1)$$

$f_2 = 1 - f_1$ and $f_1$, $f_2$ stand for the probability of memory access and disk access. Ops stands for the amount of data operation. MemTime and diskTime respectively stand for the time of a single memory access and disk access.

The number data that can be stored in the cached memory amounts to n which equals $\frac{mem}{data} * num$ . Mem stands for the size of memory space. Data stands for the size of data and num stands for the number of data. So $\frac{data}{num}$ stands for the size of a single data and we make $ave = \frac{data}{num}$ . According to the rule of zipfian and the approximate formula, we can get formula 2

$$f_1 = C + \frac{C}{2} + \frac{C}{3} + \cdots + \frac{C}{n}$$
$$C + \frac{C}{2} + \frac{C}{3} + \cdots + \frac{C}{n} \approx \ln n + c \quad (2)$$

We also know that $c = 0.577216$ in the approximating formula. What's more, we can make ave a constant by simplifying test environment. So the formula 1 and the reading throughput can be finally simplified as formula 3.

$$time = a + b * \ln mem$$
$$throughput = \frac{ops}{time} \quad (3)$$

A and b stand for all the constant which include ops, memTime, ave and so on. In the final step, we should determine the parameters a and b according to the test value shown in Table 2. In our test environment, we make ops equal 105. We use SPSS as the data simulation tool. The simulation result of parameter a and b is listed in the Table 3.

**Table 3. SPSS Simulation Result**

| | Not standard factor | | Standard factor |
|---|---|---|---|
| | B | Standard deviation | Beta |
| ln(mem) | -331.783 | 13.225 | -.998 |
| constant | 957.443 | 20.763 | |

From Table 3 we can get the conclusion that $a = 957.443$ and $b = -331.783$. So the relationship between the size of cached memory and reading throughput is expressed as formula 4.

$$time = 957.443 - 331.783 \ln mem$$
$$throughput = \frac{100000}{957.443 - 331.783 \ln mem} \quad (4)$$

This formula can explain that the increase of cached memory space can result in the increase of reading throughput. But the tendency of increase is less obvious. The data access meet the zipfian distribution law which defines that the product of probability and rank is a constant. So than more and more data will be stored in the cached memory space. But the probability of accessing those data is getting smaller and smaller which will influence the increase of reading throughput. If the memory space is infinite, we can set the size of cached memory as big as possible. But in the practical environment, the memory space is limited. So if it is set too large, it will not promote reading performance obviously any more. And on the contrary taking too much memory space will seriously influence writing performance.

The SPSS software can also calculate the deviation between the formula and the test result. Deviation image is shown in Figure 7.

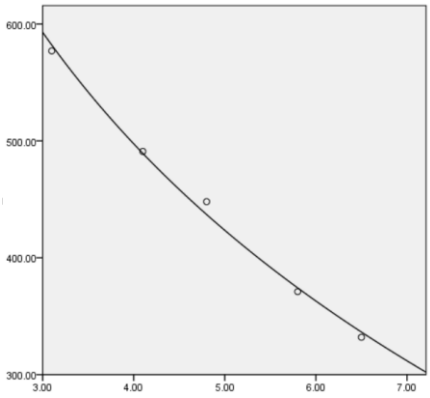



**Figure 7. Simulation and Test Results**

In Figure 7, the curve stands for the simulation result and circles stand for test results. From this image we can say that the simulation formula can match the test result very well. The detail deviation data is listed in Table 4.

**Table 4. Deviation Result of SPSS Simulation**

| R | R variance | Adjusted R variance | standard deviation of estimation |
|---|---|---|---|
| .998 | .995 | .994 | 7.730 |

### 3.2.2 The Influence of JVM Heap on the Writing Performance

From the former test we can reach the conclusion that on the one hand the reading performance of Cassandra database system will increase as the size of JVM heap being larger. But on the other hand the tendency of the increase will slow down. We will build up a more detailed relationship between JVM heap and the writing performance by further tests.

Logically, the JVM heap can be divided into Memtable space and other space which we name as Non-Memtable space. We will test those functional units separately below.

We set the size of JVM heap 2GB, the amount of data $10^7$ and the size of each data 1KB. We will finally write 10GB data and at the same time increase the size of Memtable step by step under the premise that the total amount of JVM heap is the same. The test result is shown in Figure 8.

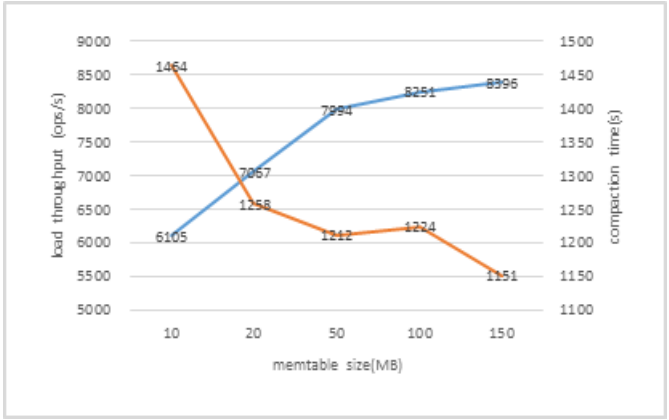



**Figure 8. Influence of the Size of Memtable on the Reading and Compaction Performance**

From Figure 8 it can be conclude that the reading throughput and compaction speed increase with the size of Memtable being larger. Next, to test the influence of Non-Memtable space we have to fix the size of a single Memtable structure to 53MB and at the same time gradually increase the size of Non-Memtable space. The result is shown in Figure 9.

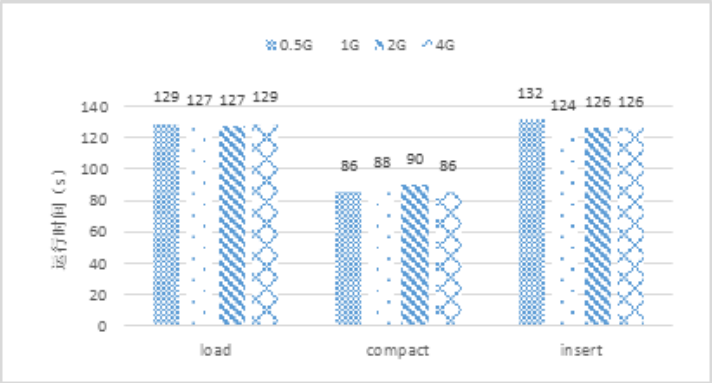



**Figure 9. Influence of the Size of Non-Memtable on the Loading, Inserting and Compaction Performance**

It is clearly shown in the Figure 9 that the Non-Memtable space have little influence on the loading, inserting, compacting performance. So we do not need to spare too much space to the Non-Memtable space. The size of the Non-Memtable space should be a constant which determined by the practical test. So we should just build up the quantitative model between the Memtable space and writing throughput.

Table 5 lists the relationship between the size of Memtable space and the time to complete $10^6$ writing operations.

**Table 5. The Relationship between Memtable Size and Time to Writing in**

| Memtable size (MB) | Memtable number | Time (s) | Deviation of Memtable number | Deviation of time (s) |
|---|---|---|---|---|
| 8.5 | 118 | 178 | | |
| 17 | 59 | 148 | 59 | 30 |
| 35 | 29 | 134 | 30 | 14 |
| 70 | 14 | 127 | 14 | 7 |
| 140 | 7 | 122 | 7 | 5 |
| 280 | 4 | 119 | 4 | 3 |

The increase of Memtable size will result in the decrease of Memtable number to writing in the disk and finally decrease the time to write data in the disk. As we multiply the Memtable size, the deviation of time is getting smaller. That is, in another word, the same as we describe before that the tendency of the increase will slow down. From the Table 5 we can also find out the proportional relationship between the deviation of Memtable number and the deviation of time. So we can suppose that the Memtable number and the total time is linearly related. Therefore, we assume that relationship can be expressed as formula 5.

$$time = a * \frac{data}{mem} + b \tag{5}$$

We firstly make $a' = a * data$ . In the next, we should determine the parameter $a'$ and b according to the test value shown in Table 5. We use SPSS as the data simulation tool. The simulation result of parameter $a'$ and b is listed in Table 6.

**Table 6. SPSS Simulation Result**

| | Not standard factor | | Standard factor |
|---|---|---|---|
| | B | Standard deviation | Beta |
| 1 / mem | 506.051 | 10.769 | .999 |
| （constant） | 118.598 | .596 | |

From Table 6 we know that $a' = 506.051$, $b = 118.598$ .So the quantitative model of the size of Memtable space and writing throughput can be expressed as formula 6.

$$\begin{aligned} time &= \frac{506.051}{mem} + 118.598 \\ throughput &= \frac{100000 * mem}{506.051 + 118.598 * mem} \end{aligned} \tag{6}$$

Deviation between the formula and the test result is shown in Figure 10.

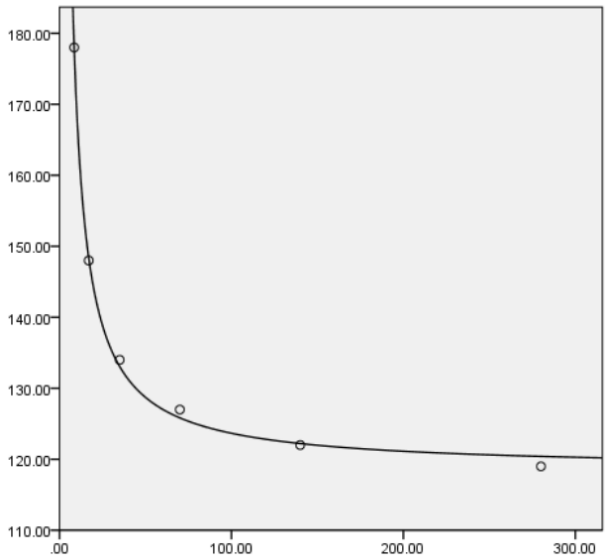



**Figure 10. Simulation and Test Result**

In Figure 10, the curve stands for the simulation result and circles stand for test results. From this image we can say that the simulation formula can match the test result very well. The detail deviation data is listed in Table 7.

**Table 7. Deviation Result of SPSS Simulation**

| R | R variance | Adjusted R variance | standard deviation of estimation |
|---|---|---|---|
| .999 | .998 | .998 | 1.053 |

For the finally simulation formula we can reach the conclusion that the writing throughput will increase with the increase of Memtable but the tendency will be less obvious. This characteristic is the same as the result test before.

However, if the size of Memtable is too large, it will obstruct the promotion of system performance. In another word, the size of Memtable has its own limitation to promote system performance. The test result is listed in Table 8. We set $10^6$ writing operation, 16GB memory space and 8GB JVM heap space.

In contrast, the setting of 400MB Memtable will achieve the best performance. Firstly, the deviation of the number of disk writing will decrease with the increase of Memtable size. This will make the decrease of the time less obviously. Secondly, we can guess from the characteristic listed in the Table 8 that maybe flush too much of the data into disk a time will cause the obstruction of writing queue.

**Table 8. Performance Characteristic of Different Memtable Size**

| Size of Memtable | time（s） | throughput（ops/s） | Average throughput(ops/s) | Characteristic |
|---|---|---|---|---|
| 200MB | 350 | 2857 | 2792 | Every 20 second, the throughput will decrease to several hundred or one thousand from ten thousand, and it will last about 10 seconds |
| | 366 | 2732 | | |
| | 359 | 2786 | | |
| 400MB | 336 | 2976 | 2911 | Every 40 second, the throughput will decrease to several hundred or one thousand from ten thousand, and it will last about 20 seconds |
| | 344 | 2907 | | |
| | 351 | 2849 | | |
| 800MB | 357 | 2801 | 2847 | Every 70 second, the throughput will decrease to zero, and it will last about 20 seconds. |
| | 348 | 2874 | | |
| | 349 | 2865 | | |

In this situation, it will decrease the throughput and at the same time will make users wait for a long time.

So we test on the writing speed of the disk. The result is shown in Figure 11.

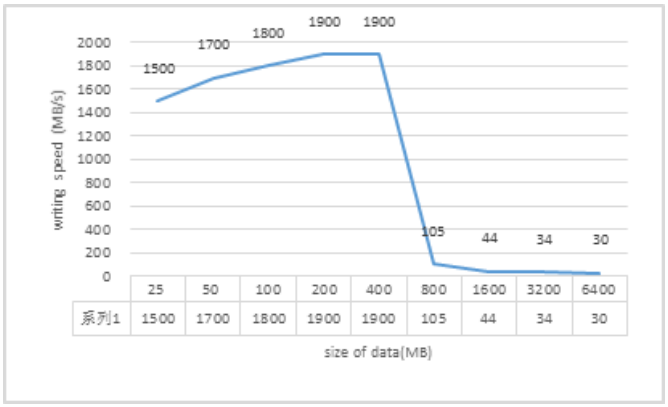



**Figure 11. The Relationship between the Size of Data and the Writing Throughput**

At first the writing speed slightly increases when the size of data become larger. But when the size of data is between 400MB and 800MB, the writing speed sharply decreases. Finally the speed stabilizes at a low value. This result could well explain the characteristic listed in the table 8. We could conclude that the obstruction of writing queue finally results in the decrease of writing throughput and writing obstruction. When the amount of data flushed into disk a time reaches the limitation, it will seriously influence the writing performance and make the next writing operation more clogged.

So as to the formula 6, the parameter of mem should have its own limitation which is related to the disk writing performance. When the limitation is exceeded, the formula could not be applied anymore and will also make the writing throughput fluctuate and decrease.

## 4. Memory Allocation Model in Practical Condition

A column family corresponds to a sole Memtable structure. So the number of column family equals to the number of Memtable structure. When the Memtable structure reaches the limitation, it will be put into the disk flush queue and finally flushed into the disk synchronously or asynchronously. The maximum number of the threads that operate the disk writing process could be set as the number of disks.

Just as we discuss before, we could set the limitation of the number of data that will not occur the decrease of disk performance as $max_{write}$. In each server the average number of column family is set as $n_{cf}$. According to the characteristic of server and application, we can set the Non-Memtable space a constant namely $mem_{non\_memtable}$. The former formulas and the memory allocation can be summed up as formula 7.

$$\begin{aligned} throughput_{write} &= \frac{ops_{write} * mem_{memtable}}{a_1 + mem_{memtable} * b_1} \\ throughput_{read} &= \frac{ops_{read}}{a_2 + b_2 * \ln mem_{cached}} \\ mem_{total} &= mem_{memtable} + mem_{non\_memtable} + mem_{cache} + mem_{sys} \end{aligned} \tag{7}$$

The $ops_{write}$ and $ops_{read}$ separately stand for the number of reading and writing operation. The a1、b1、a2、b2 are the parameters that can be determined by the SPSS simulation. The $mem_{cached}$ stands for the space of cached memory space and $mem_{memtable}$ stands for the space of Memtable space. The $mem_{sys}$ is a constant which is related to the system and the other applications running on it.

If the Memory space is infinite, to reach the best writing performance, we must set $mem_{memtable} = n_{cf} * max_{write}$ and $mem_{non\_memtable}$、$mem_{sys}$ to be constant. The other space could be used as $mem_{cached}$ to realize a better reading performance. But in practical

condition, the memory space is limited. If we allocate the memory space according to the method above, the Memtable structure will take too much space that will in turn influence the reading performance. Besides the $mem_{non\_memtable}$ and $mem_{sys}$ which could be set to be constant, the remaining memory space could be expressed as $mem'_{total}$ and $mem'_{total} = mem_{memtable} + mem_{cache}$. So the formula 7 could be expressed as formula 8. And the size of $mem_{memtable}$ has the limitation of $n_{cf} * \max_{write}$.

$$\begin{aligned} throughput_{write} &= \frac{ops_{write} * mem_{memtable}}{a_1 + mem_{memtable} * b_1} \\ throughput_{read} &= \frac{ops_{read}}{a_2 + b_2 * \ln mem_{cached}} \\ mem'_{total} &= mem_{memtable} + mem_{cache} \end{aligned} \tag{8}$$

Assume that we have to meet the need of reading and writing performance which we set as $read_{max}$ and $write_{max}$ as far as possible. Then we need to trade off the allocation of memory space. Firstly we need to assign a weight to the importance of reading and writing performance which we can express as $rate_{read}$ and $rate_{write}$. The final quantitative performance could be expressed as formula 9.

$$Perf = rate_{read} * \frac{throughput_{read}}{read_{\max}} + rate_{wirte} * \frac{throughput_{write}}{write_{\max}} \tag{9}$$

Secondly, we could reach the best performance in the range of valid data according to formula 8 and formula 9.

## 5. Conclusions

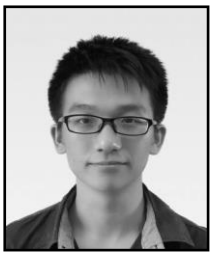
In this article, firstly we find out the certain functional units of memory space that will influence the system reading and writing performance according to the analysis of Cassandra database operation process. Secondly, build up the quantitative model of the relationship between the size of memory space and the reading and writing throughput in the condition that the memory space is infinite. We can conclude from the Figure and data result of deviation that this model could simulate the system performance very well. Finally we improve the model by applying it in the practical condition. This quantitative relationship model could help us to find out a best solution according to the actual demand.

## References

[1] M. Ferdman, A. Adileh and O. Kocberber, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware", ACM SIGARCH Computer Architecture News, vol. 40, no. 1, **(2012)**, pp. 37-48.
[2] P. L. Kamran, B. Grot and M. Ferdman, "Scale-out processors", ACM SIGARCH Computer Architecture News. IEEE Computer Society, vol. 40, no. 3, **(2012)**, pp. 500-511.
[3] First the tick, now the tock: Next generation Intel microarchitecture (Nehalem). White Paper, **(2008)**.
[4] T. Rabl, M. Sadoghi, H.-A. Jacobsen, S. G. Villamor, V. M. Mulero and S. Mankowskii, "Solving Big Data Challenges for Enterprise Application Performance Management," PVLDB, **(2012)**.
[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-Value Store," in SOSP, **(2007)**.
[6] R. Cartell, "Scalable SQL and NoSQL data stores," SIGMOD Record, **(2010)**.
[7] T. Nguyen and M. H. Nguyen, "Zing Database: high-performance key-value store for large-scale storage service", Vietnam Journal of Computer Science, **(2014)**.
[8] "The Apache Cassandra Project", http://cassandra.apache.org/.
[9] C. Chen and M. Hsiao, "Bigtable: A distributed storage system for structured data", Proceedings of Osdi', vol. 26, no. 2, **(2006)**, pp. 205-218.

## Author



**Boqian Wang**, born in 1990, graduated student, major in computer science and technology. Main research area is computer architecture and big data application, especially in Cassandra database.