Use of Synchronized Context Broker Cache In Cloud

Bipin B. Shinde\textsuperscript{1} & Dr. Richa Gupta\textsuperscript{2}
\textsuperscript{1}(PG Student) Department of Computer Science and Engineering, Sushila Devi Bansal College of Engineering, Indore
\textsuperscript{2}(Head Department) Department of Computer Science and Engineering, Sushila Devi Bansal College of Engineering, Indore

Abstract: Performance is key factor for every good application. There are number of ways by which application performance can be increases. Context aware system is one of the application which uses context information. Many of them are using broker based architecture. Contextual data cannot alter frequently; it will remain valid for certain period. Hence Caching mechanism can impressively uses to access variety of context data very quickly. Considering these two factors, first use of improvised broker architecture and second established cache mechanism applying on broker cache will definitely improve the performance of system. This paper focuses on synchronised context broker cache and use cache mechanism in cloud environment. Using synchronised context broker cache and caching strategies efforts are made to utilise data in different context broker cache with synchronously so that within short period of time it will be available to user and overall application performance will be improve. The results shows advantages of using Synchronised broker cache over asynchronies broker architecture with caching mechanism. The relevance of synchronies broker caching mechanism is prototyping in cloud environment.

Introduction

In today’s scenario cloud computing is most widely used technology over internet. Cloud computing provides different types services mainly in software based, platform based and infrastructure related. It offers scalability, reliability, performance and interoperability to the internet user. Performance is key factor for every system and so as to cloud computing. Context ware System is one of service which is use as Software as Service application in cloud computing. For context aware system cloud is ideal platform to represent complete usefulness. Context aware system is used to serve contextual information about humans and computational aspect. Contextual data includes personal situation, motion and acceleration, physical and environmental attributes. Context aware system performs different functions like acquiring context information, context storage, and abstraction levels of context and distribution of context information. Many software services used server/broker model such as context aware system. In this system broker is intermediate part between server and client. There are number of way by which cloud performance can be increase. One of the methods is to use cache techniques.

Applying suitable established cache mechanism can lead to provide improves performance results. This server/broker model can be deployed in cloud environment with established cache policy in order to get better performance. Existing context aware system in cloud has not utilized cache mechanism and synchronised broker effectively.in order to make wide spread use of context aware system in cloud computing is huge platform which can offers performance, reliability and scalability. Yet not seen combination of context aware system and cloud.

In cloud computing when software as service is consider many service providers are there which uses server/broker based architecture. Context aware system is an application service run on cloud in the form software as service. This model consists of client, Broker and provider. Client is user sending request data which handles by broker present in cloud. Broker is an added service that extends the power and performance of system.it consisting of data cache of limited memory size. It contains recently respond requested data ,if new requested data is found in cache item list then immediately send to client otherwise forwarded to provider ,further provider lookup in its database and send back to broker. Broker in cloud updated its cache item list and notify to client. Contextual data has its own property that it can valid for specific time period hence caching mechanism can effectively apply on contextual data and gives significant results. There have been many efforts taken to improve the performance of cloud services as it is used by user. Considering these factor this paper gives an attempt to improve the performance of cloud application.
such as context aware system by using synchronize broker cache architecture with cache mechanism. Brokers are residing in cloud network each having its own cache. When user send query request, it will arrive randomly to one of broker. Here in cloud number of broker will be there and their cache having different contextual data serves previously. For making utilization of broker cache data, broker cache need to synchronize. Using synchronised broker cache and suitable cache mechanism will give significant improvement over asynchronies broker cache. Once such effectiveness is established through experimental analysis, we analyse and compare the performance of the synchronised broker cache with asynchronies broker cache with caching strategies in a prototype Cloud-based environment.

RELATED WORK

A different server/broker-predicated systems have been developed, e.g. CoBrA [2], SOCAM [3], JCAF [4], PACE [5], and MobiLife [6] but no one has utilize caching mechanism efficaciously. The MobiLife designates context caching at the context provider component but this approach engenders distributed context caches at each context provider, potentially preserving computational load at the providers but not reducing the communication cost. The query from the client has to travel from the context provider via the context broker which increases round trip time. This mechanism can be altering by using a collective cache at provider level. Lots of study are done on cache management strategies and its use in cloud computing. Yu Chen et al [7] provide research cognate to mobile users to access location predicated accommodations but for this purport mobile users has to disclose its location and for security reason it is vulnerably susceptible. This paper provides solution for privacy aegis by utilizing the K-anonymity model. However, the computational and communication cost is high. This research proposes cache management techniques for amendment of utilizing privacy aegis and preserving computational potency, and decrementing communication costs. Nicolas Le Scouarnec et al. [8] examine several techniques of cache mechanism for cloud. To deal with this new context, provide design and evaluate a new caching policy that minimizes the cost of a cloud based system. The policy takes into account the frequency of consumption of an item and the cloud cost model. Lakshmish Ramaswamy et al. [9] study different challenges in designing a large scale cooperative edge cache network. Introduce the concept of cache clouds, which forms the fundamental framework for cooperation among caches in the edge network. Gregory Chockler et al. [10] discuss about data cache as cloud services. Research is to provide data cache with a simple multi-tenant data cache. It is based on clock replacement Algorithm which is offer Minimum QoS-Guarantees, Maximize Overall Hit Rate and Maximize Concurrency. Si Changing [11] proposes studies of previous cache management technology of cluster or distributed system, and puts emphasis on availability and consistency, which bring down availability and scalability of cloud storage system. This also include analyses the adaptability of previous cache management technology for cloud storage environment. Some people have focused the importance of caching context information but no study work has reported any results. Saad Liaquat Kiani et al. [12] have done impressive work and shown how use context cache with different caching mechanism lead to get improves performance of system. Their work is main motivation for continuing study of this domain. In this research context broker architecture is considered but without synchronisation of broker cache. Context consumers request context data about a particular entity and scope by forwarding a Context encoded query to the context broker. The broker forwards the query to an appropriate context provider that can satisfy it. When the query-satisfying context information is available, the provider sends the context response to the broker. In the absence of a caching facility, the broker simply forwards the query to the querying consumer. The Context Provisioning Architecture utilizes a caching component that caches recently received contextual data in response to context queries, in addition to forwarding the response to the querying consumer. The context data remains in the cache for the validity period unless it is replaced by more recent context of the same scope/entity or has to be removed to free the cache due to cache size limits.

1. SINGLE SERVER/BROKER ARCHITECTURE

The single server broker architecture simply based on client-server model in which context information related services request by user or client and serve by server. These component entities interconnected by broker that provide query satisfaction and lookup services. This architecture consists of three main component context user or client that utilises context information. User can retrieve context information by sending a subscription to the Context Broker or a direct on-demand query and broker will delivered context information when it is available. The Provider or server database component provides contextual information. A server gathers data from a collection of sensors, network/cloud services or other relevant sources. Fig. 1 single server/broker architecture depicts context client, broker and server database or provider.
2. ASYNCHRONIES BROKER CACHE IN CLOUD

The number of architecture of cloud broker has been proposed. Some architecture proposed context cache in cloud and discuss the importance of caching context in improving its performance. Few context broker architecture is considered but without synchronization of broker. First Client will register to particular context broker in cloud. Every Broker is uses cache having size up to 10 requests. Every broker will register with particular context provider. Client will send request, according to client_id request go to specific context broker if data available in broker cache then it will replay to client. If data not available in the cache of broker then sync cache algorithm try to find it in other broker cache of broker cloud. If search is successful requested data is send to broker and the broker will replay to client. In case request is new and data is not available in Broker’s Cloud query will forward to appropriate provider then the provider will look up and replay to broker. The broker will update his cache and then replay to client. In this scheme mean query satisfaction time when query is not present in broker cache is always greater than the query present cache. Suppose random queries executes on this architecture then comparing result of mean query satisfaction it is observed that T_x is time for processing one query then mean query satisfaction time will be formulated by following way and shows relationship that

$$\sum T_{sc} > \sum T_{cs}$$

(1)

Here in equation (1) $T_{sc}$ is mean query satisfaction time of processing query when it is not present in broker cache and it is greater than $T_{cs}$ mean query satisfaction time of processing query when it is present in broker cache.

SYNCHRONISE CONTEX BROKER CACHE ARCHITECTURE

In server/broker model federation of broker in cloud is as shown in figure 1. One of way to achieve this by synchronising the brokers in cloud. In synchronised broker architecture all brokers present in the cloud are synchronised to each other. Here meaning of synchronising is that broker cache should connected to each other. This paper proposes synchronized context Broker cache model. Each broker in cloud has its own cache and virtually it can be connected to each other. In this way the size of cache increases and there is more probability of cache hit. Also the mean query satisfaction time get reduce. When client sends query to any broker in cloud first it checks in its cache, if there is cache hit then query can immediately serve otherwise for cache miss instead of forwarding that query to provider it can check in other brokers cache randomly. So many brokers are available in cloud hence probability of cache hit is more. Hence the time period to serve query through synchronised broker cache is less than to satisfy with synchronised broker.

Synchronizing Broker cache architecture is as shown in Figure 2. Each broker in cloud has its own cache and virtually it can be connected to each other. In this way the size of cache increases and there is more probability of cache hit. Also the mean query satisfaction time get reduce.

When client sends query to any broker in cloud first it checks in its cache, if there is cache hit then query can immediately serve otherwise for cache miss instead of forwarding that query to provider it can check in other brokers cache randomly. So many brokers are available in cloud hence probability of cache hit is more. Hence the time period to serve query through synchronised broker cache is less than to satisfy with synchronised broker. Mean query satisfaction time using synchronized context broker cache shows different results. Consider an
The experimental setup consists of four brokers each having a cache size limit of 10 queries. Brokers are communicated with each other by the syn_broker algorithm. Some set of queries executed under this condition. Query processing time for broker B1 is \( T_{x1} \), similarly for remaining brokers B2, B3, B4 it would be \( T_{x2}, T_{x3} \), and \( T_{x4} \) respectively. Results can be put mathematically to compare query satisfaction time when queries are processed from a provider’s database and when the query is processed from synchronized context broker cache. The satisfaction time taken by a provider database is always greater than the time taken from to satisfy query from synchronized context broker cache.

Consider that query arrives from a client and processes in any broker in the cloud. If requested query data is searched in broker B1 cache, then it will be required delay i.e. searching time say \( T_{x1} \).

\[
B1 = T_{x1}
\]  

(2)

If data is not found in Broker B1 Cache then it will be searched in Broker B2 Cache and it also required delay say \( T_{x2} \) and total delay is.

\[
B2 = T_{x1} + T_{x2}
\]  

(3)

If data is not found in Broker B2 Cache then it will be searched in Broker B3 Cache and it also required delay i.e. processing time say \( T_{x3} \) and total delay is.

\[
B3 = T_{x1} + T_{x2} + T_{x3}
\]  

(4)

If data is not found in Broker B3 Cache then it will be searched in Broker B4 Cache and it also required delay i.e. processing time say \( T_{x4} \) and total delay is.

\[
B4 = T_{x1} + T_{x2} + T_{x3} + T_{x4}
\]  

(5)

Now consider query satisfaction time to process requested data from providers database is \( T_p \), then conclusion can be made that query satisfaction time period \( T_p \) for processing query from providers database is always greater than the query satisfaction time of synchronized context broker cache.

\[
T_p > (T_{x1} + T_{x2} + T_{x3} + T_{x4})
\]  

(6)

From equation (5) prediction can be drawn that in a federation of context broker cache if they are synchronized with probability of finding requested context data in one of cache increases. Total mean query satisfaction time for context data traversing through brokers and then provider to satisfy query always greater. This equation theory we are trying to prove with the help of experimental set up.

**PROPOSED ALGORITHM**

One of the main advantages of synchronization of broker is that cache size is increase virtually. Because all broker cache in cloud connected to each other. Syn_broker cache algorithm allows communicating with each broker cache before query forwarding to provider. The query processing and Broker Synchronization operations from the synchronized broker’s point of view are described in Algorithms 1 and Algorithm 2.

**Algorithm 1**

Client Broker Query Processing

Where \( P = \{p1, p2, p3, \ldots\} \)

is set of providers.

Where \( Q = \{Iq, Tq\} \)

\( Iq \) is Query ID & \( Tq \) is Query Arrival Time.

Where \( B = \{B1, B2, B3, \ldots\} \)

is set of Broker in cloud.

Subscribe (Q) # Query arrives at broker randomly and stores in broker table.

\( Q = \) random \( \{B1, B2, B3, \ldots, B4\} \)

Record (Iq, Tq)

Cq Broker CacheSearch (Ie, Iq, Is, Bid)

# searches in Broker cache.

If Cq

then

Notify (Cxc, Cqs)

Notify the client as cache hit.

IncrementUseCount (Cqs)

else

Cqs = Syn_BrokerCacheSearch (Bid, Ie, Q, Is)

# searches in next broker cache

Notify (Cxc, Cqs)

# notify the client as cache hit.

IncrementUseCount (Cqs)

else

Pq = lookup (P, Ie, Is)

# broker lookup for suitable provider.

Query (Q, P)

# Query forwarded to provider and searches in providers lookup.

**Algorithm 2**

Syn_BrokerCacheSearch Algorithm

Where \( B = \{B1, B2, B3, \ldots\} \)

is set of Broker in cloud.

Subscribe (Q) # Query arrives at broker randomly and stores in broker table.

\( Q = \) random \( \{B1, B2, B3, \ldots, B4\} \)

Record (Iq, Tq)

Cq = Syn_BrokerCacheSearch (Bid, Ie, Q, Is)

# searches in next broker cache

If Cq

then

Notify (Cxc, Cqs)

Notify the client as cache hit.

IncrementUseCount (Cqs)

else

Pq = lookup (P, Ie, Is)

# broker lookup for suitable provider.

Query (Q, P)

# Query forwarded to provider and searches in providers lookup.
Else If Cqs in B2 then
Notify (Cxc, Cqs)
#increment the use count of Query.
IncrementUseCount (Cqs)
Notify the client as cache hit.
If Cqs in B3 then
Notify (Cxc, Cqs)
#increment the use count of Query.
IncrementUseCount (Cqs)
Notify the client as cache hit.
If Cqs in B4 then
Notify (Cxc, Cqs)
#increment the use count of Query.
IncrementUseCount (Cqs)
Notify the client as cache hit.

RESULT

Every Client is register to particular context broker in cloud. Assuming that broker is uses cache having size up to 10 requests. Every broker will register with particular context provider. Some set of group of queries performed with oldest first cache policy using asynchronous broker caching policy. Client will send request, according to client_id request go to specific context broker if data available in broker cache then it will replay to client. If data not available in the cache of broker then query will be forwarded to appropriate provider because brokers cache are not synchronised. Further the provider will look up and replay to broker. The broker will update cache and then replay to client. In this case round trip time of requested query increases. Query is to travers from broker to provider then again to broker and finally to client. This set up is established and experiment conducted in cloud environment. Group of queries will be performing on set up which consist of Client, asynchronous broker cache and provider. Analysis is done by considering reference case no cache show maximum performance improvement with this setup.

Using synchronisation algorithm same experiment is conducted for synchronised broker cache with oldest first cache policy and observe the result for same. Client will send request, according to client_id request go to specific context broker if data available in broker cache then it will replay to client. If data not available in the cache of broker then sync cache algorithm try to find it in other broker cache of broker cloud. If search is successful requested data is send to broker and the broker will respond to client. in this case main advantage is that due to synchronisation of broker cache probability of getting requested query in broker cache increases. Further different caching strategies may give more effective results. Analysis is done and result is considered for comparison. Table 1 shows average mean query satisfaction time to process query. Processing time calculated in milliseconds.

Table 1 Comparison of Processing Time of Synchronous and Asynchronous Context Broker Cache

<table>
<thead>
<tr>
<th>Set of Queries</th>
<th>Processing time for Synchronous Broker(ms)</th>
<th>Processing time for Asynchronous Broker(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>260</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>240</td>
<td>258</td>
</tr>
<tr>
<td>300</td>
<td>170</td>
<td>250</td>
</tr>
<tr>
<td>400</td>
<td>150</td>
<td>360</td>
</tr>
<tr>
<td>500</td>
<td>225</td>
<td>402</td>
</tr>
<tr>
<td>600</td>
<td>110</td>
<td>208</td>
</tr>
<tr>
<td>700</td>
<td>210</td>
<td>247</td>
</tr>
<tr>
<td>800</td>
<td>230</td>
<td>293</td>
</tr>
<tr>
<td>900</td>
<td>174</td>
<td>271</td>
</tr>
<tr>
<td>1000</td>
<td>172</td>
<td>230</td>
</tr>
</tbody>
</table>

Both results are compared for proving effectiveness of performance of cloud based synchronized context broker architecture. The comparison is done for experiment conducted for fixed size of cache limit using oldest remove first cache policy. It is observed that average mean query satisfaction time for synchronized context broker cache architecture is 176(milliseconds) whereas for asynchronous context broker it is 280(milliseconds). the performance improvement can be clearly seen by use synchronized context broker cache. The comparison results are plotted in figure 5.

![Fig. 3 Result of Comparison of Synchronous and asynchronous broker cache.](image-url)

For more effective result of mean query satisfaction time with synchronized context broker cache model different well established caching policies are considered. Same set of group of queries are performed to analyzed mean query satisfaction time. In this experiment three caching policies Least used (LU), Soonest Expiry First (SEF) and Randomly remove are used. In Least used cache replacement policy item is remove which has been
retrieved less number of time. In SEF policy some fix validity time is given to arrive query in cache and according to that an item form cache being removed closest to its validity expiration and in randomly remove policy an item is remove randomly without any conditioning. Figure 4 shows comparison of average mean query satisfaction time for synchronized context broker cache using different caching policies.

![Fig 4 Result of Average Mean Query Satisfaction Time with different Cache Policies](image)

Results shows fairly distinct performance when synchronized Context broker model used with different caching insertion and removal policies. In case of OF and LU caching policy performance remain constant with very marginal result (198 ms for OF and 200 ms for LU) whereas SEF policy become more effective for context data having long validity (195 ms). Random removal cache policy shows significantly same performance as SEF policy (195 ms) for most type of context data. Since context data consist of verity of range further it can be possible to apply specific caching mechanism and insertion and removal policies accordingly type of context query.

CONCLUSION AND FUTURE WORK

The work presented in this paper builds on synchronised context broker cache and the well-established mechanism of caching in distributed systems However, the use and effectiveness of caches has limitation and not evaluated. Synchronization of context broker cache is effective way to utilise other broker cache in cloud architecture for searching requested query, which definitely improve the mean query satisfaction time between client and providers. In this paper focus is on improvement of performance of cloud based system by using synchronised context broker cache and cache mechanism.

In order to improve performance of cloud using cache mechanism two most important things are consider first one size of cache and to keep useful data in cache best insertion and removal policies. This factor certainly makes a difference to the performance of any application. In this paper synchronization of various brokers in cloud would help to increase the size of cache. Form result it is observed that average mean query satisfaction time for provider database is always greater than the query satisfaction time for synchronized broker cache. To prove this through experiment the mean query satisfaction time for group queries have been tested and comparison between use of synchronized context broker cache architecture and asynchronous context broker cache shows greater difference in mean query satisfaction time. In the future cloud based broker architecture can be more improve to provide better performance. There is scope to apply different cache policy accordingly type of context being used and observe effective results. In the future more effective study will be needed to sort different type of context data and then applying suitable cache policy. Different modified server broker architecture can be proposed and various cache strategies will be applied according to suitability of context data. Further study and research to development of this domain will be expected.

References


In Workshop of the HP OpenView University Association.


