

Case study: Challenges faced by Rakuten in achieving performance improvement and flexible architecture using DCP

While there is rapid globalization in many Japanese companies, there is a demand for a transformation of the application structure itself in order to cope with growing business needs. Rakuten is striving the construction of a flexible application environment, that on one hand supports system work load that follows a sudden expansion of business it is currently experiencing, and also lives up to future business needs that are expected to diversify with global development, by combining the distributed caching platform (DCP) with loosely-coupled multilayer application structures.

Gartner

© 2011 Gartner, Inc. and/or its Affiliates. All Rights Reserved. Reproduction of this publication in any form without prior written permission is forbidden. The information contained herein has been obtained from sources believed to be reliable. Gartner disclaims all warranties as to the accuracy, completeness or adequacy of such information. Gartner shall have no liability for errors, omissions or inadequacies in the information contained herein or for interpretations thereof. The reader assumes sole responsibility for the selection of these materials to achieve its intended results. The opinions expressed herein are subject to change without notice.

Summary

Rakuten (see Remark 1) has 2 categories of customers; business operators who provide services to general consumers who use internet business services, such as shopping malls (Rakuten Inc.), and operators who provide services to general consumers who use their internet service platforms. As measures to cope with future needs, Rakuten was seeking solutions for both categories of customers in terms of performance improvement in response to future increase in web transactions, and freedom from limitations in utilization time that accompanies database (DB) maintenance operations. As a solution to these problems, Rakuten made stepwise efforts with the objective of constructing a flexible application structure that gives better performance and enables prompt addition and modification of services in future. Rakuten decided on a loosely-coupled, multilayered service oriented architecture (SOA) and introduced DCP (See Remark 2) as the implementation technology for this architecture, thereby completing their 1st step towards transformation of application structure with flexible architecture, and achieved definite results.

Major observations

- DCP is effective in response improvement in terms of data access requests from user interfaces (UI) such as large-scale web etc., while at the same time maintaining transaction consistency.
- DCP is effective in restraining the effects or limitations of utilization time generating from operation interruptions such as DB maintenance, thereby avoiding loss of business opportunities.
- There are actual case examples to substantiate the use of DCP in materialization of SOA reference architecture having loosely-coupled multilayered structures.
- On the other hand, DCP also has products with unique features and reconstruction of existing applications may be required for its adoption. Therefore, along with confirmation of required migration expenses, attention must also be paid to an application design that avoids dependence on specific products.

Recommendations

- DCP can be added to the options as an effective problem-solving approach when it is necessary to achieve freedom from limitations in utilization time that accompany improvements in transaction performance or DB maintenance operations while at the same time sustaining a full-width increase in the number of users.
- DCP is not just an effective means of resolving bottlenecks in DB access, but also as an effective approach that can be used for materialization of application structures equipped with continuous and flexible performance improvement abilities, by combining design criteria of loosely-coupled, multilayered structure architectures (SOA).
- Use of DCP increases probability of successfully meeting the business needs requiring large-scale processing requests from front ends such as, enterprises with increasing number of users per single application owing to global scale system consolidation, private clouds etc. It is worthy of consideration as an installation technology that must be included in future IT roadmaps.

Overview

In order to meet future demands for prompt services and continuous performance amplification in terms of applications offered to general consumers and service operators, Rakuten took a new approach on the structural side, instead of opting for reinforcement of existing expensive hardware or parallelization of simple cheaper version hardware. Under its future plans regarding materialization of application structures with outstanding flexibility, Rakuten adopted SOA as the architecture and introduced DCP products as the implementation technology so as to obtain a loosely-coupled, multilayered system structure. As a result, UI and application logic could be separated, and with the construction of a structure with little relation between application and DB, overall performance amplification and better availability could be achieved, with a margin for future performance amplification. Moreover, they acquired development abilities for improving development productivity and development of flexible and prompt services for heterogeneous clients.

Case study:

Challenges, status quo, and background of initiatives

Rakuten, a company operating the maximum internet shopping malls in Japan, has 2 types of customers. First category includes general consumers who use the company's internet business services, and the 2nd category includes operators (service operators) who provide diverse services to general consumers using the company's internet services platforms (See Fig. 1). Rakuten was facing the following challenges in terms of its respective clients.

Services for general consumers

- Deterioration in response to increased orders from the Web (necessity of improved performance and structural improvement)
- High cost of application maintenance and repair (necessity for structural improvement)

Services for service operators

System orientation is the foundation of service management (individual services, shared services) for service operators. The challenges faced by Rakuten in system orientation included the following;

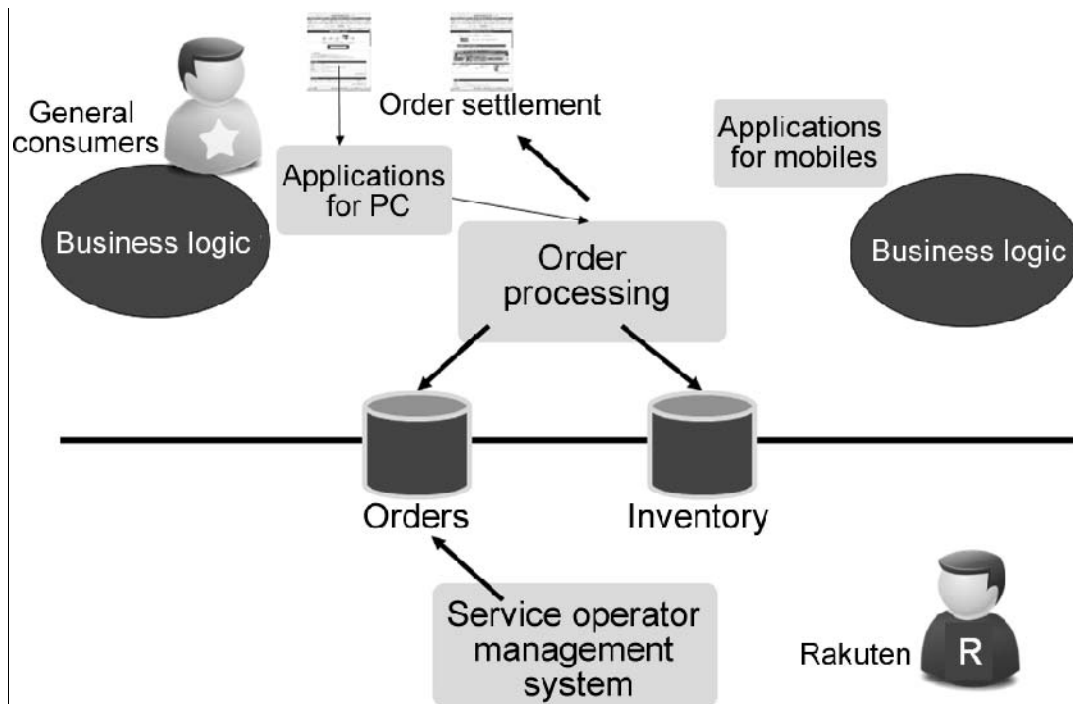
- Deterioration in response to increased orders from the Web (necessity of improved performance and structural improvement). Common service and system problems faced by general consumers.
- High cost of application maintenance and repair (necessity for structural improvement). Common service and system problems faced by general consumers.
- The system was unable give prompt responses to user needs since data for different purposes was managed on a single DB.

Apart from these it was essential to deal with the following internet service business related challenges from future point of view.

- Cheaper solutions for web transaction volume or data volume that would continue to increase in future.
- Flexible responses to consumer needs, early stage introduction of new functionalities (necessity of structural improvement)
- A system wherein services are not affected in case of a malfunction (necessity of structural improvement)
- Seeking 100% availability, high quality (necessity of structural improvement). Service continuity at the time of planned shutdown of back-end DB.

These problems were not really serious and were in fact considered to be systematic measures for preventive handling of potential risks to services sensed by Rakuten in terms of future business expansion. Particularly, since such situations have a direct impact on internet sales, Rakuten concluded that there was a need for taking early preventive measures, and took mid-term valid measures by opting for a systematic approach instead of haphazardly handling the situation.

Fig. 1 Outline map of service applications for future consumers



Source: Created by Gartner based on Rakuten documents

Problem-solving approach

As mentioned above, Rakuten fixed their internal problems and recognized the necessity of a radical structural improvement for dealing with future expansions apart from their present main objective of performance improvement. They adopted the following approach after investigating different solutions to the problems.

1.2 Discontinuation of 2-phase commit

It is necessary to maintain consistency between order information and inventory information for appropriate processing of orders from general consumers. Since Rakuten maintains this information in 2 separate databases, it had adopted a 2-phase commit system for maintaining consistency between 2 databases in realtime. However, applications also become more and more complex as the business expands. Apart from this, access to order information is not only required for processing the customer orders, but also by the service operator system for arranging for products. This increased the load on DB, and maintaining consistency in realtime with 2-phase commit became very problematic in terms of performance. Accordingly as an alternative measure for ensuring consistency of both databases, they constructed an asynchronous structure, wherein they separated the order DB and the inventory DB to remove their interdependence. This judgment was based on the fact that service operators do not process information on products purchased by user in real time, therefore, a delay of several minutes is allowed even if there is a time lag in information update in order DB and inventory DB.

More specifically, consistency between 2 databases was ensured by narrowing down the DB holding data, for which consistency needs to be maintained unconditionally in real time, into 1 (inventory DB), and then distributing update information of inventory DB to order DB using asynchronous caching. As for the process flow, data for retrieval is generated simultaneously while updating inventory information when the order is placed, and the application side is informed about its completion. If the DB is operating, order data is immediately reflected in the DB. Even though latency is generated, this data is indisputably reflected even if the DB is down or there is high load on the DB.

This was decided on the basis of the following idea. The problem of consistency arises only between inventory in the system and the actual inventory. Due to this reason, inventory information is managed as a real time transaction at the time of order settlement. Once inventory information is updated, it must be consistent with the order information under all circumstances, and it must also get reflected in the order information. Therefore as a precautionary measure, the retrieval data is maintained even when data related to queue inventory is lost. Queue is restarted if there is a problem in it and order DB can be updated by releasing restored data to the queue. Order processing by service operators may not necessarily completed in real time at the same moment when order is received, with there being some (up to a few minutes) delay. However, this does not cause any practical problems. Moreover, data reflection in DB may get delayed by a few hours if there is any trouble in the DB, but still it was determined that safe accumulation of order information without any influence on the general consumers prevents loss of business for the service operators.

2. Elimination of dependence of UI and application logic on DB

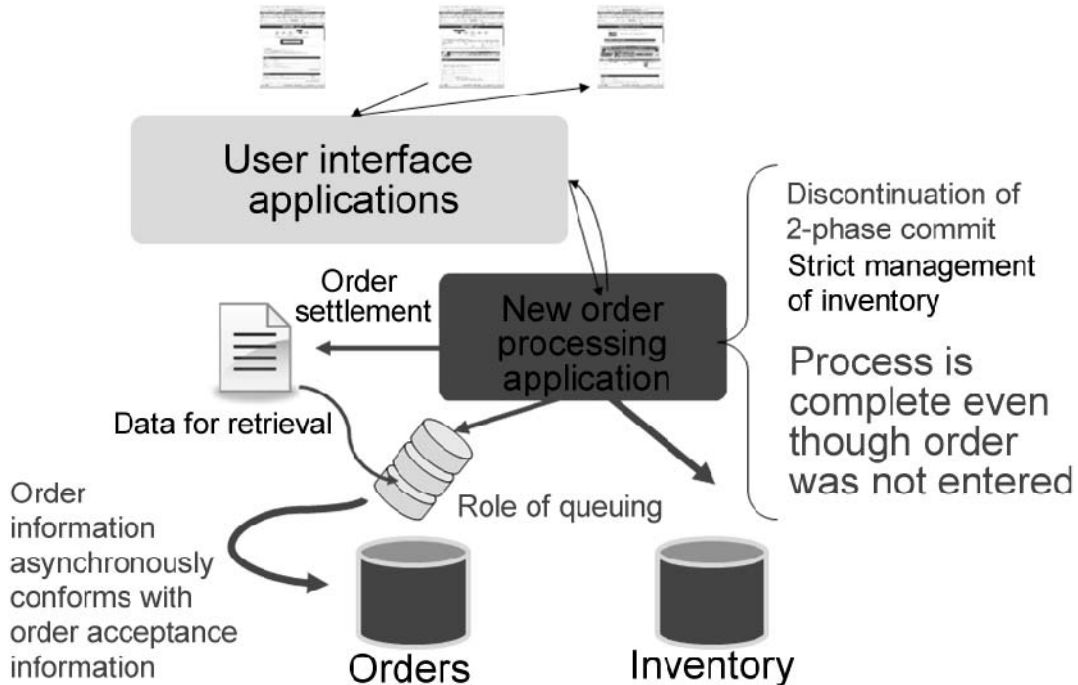
Load on the DB storing order information increases due to order processing from general consumers as well as from the service operators, which leads to a situation where general consumers choose not to make the purchase, and this needs to be avoided. As a solution to this problem, Rakuten separated order DB and inventory DB from UI and application logic as they thought that it is not only important to improve the data access performance, but also to eliminate the dependence on DB.

3. Uniform management of common business logic of individual applications and individual processing logics

- Management of essential processes such as order processes, inventory processes, user profile processes was centralized by consolidating them at a single location. However, the processes of each user (general consumer), service operator, and service must be implemented separately.
- There should be no interdependence between display requirements of UI application and individual business requirements. This increases re-usability of the UI application.

Fig. 2 gives an outline on service applications reflecting the aforementioned approach

Fig 2. Outline map of service applications for consumers having improved versions



Source: Created by Gart

Selection of technology for specific system structures: Deciding on distributed caching

At first, Rakuten considered the 2 standard approaches given below;

1. Simplification of system with loosely-coupled, layered division
2. Cost optimization

1. Simplification of system with loosely-coupled, layered division

In the existing applications there was strong interdependence of UI layer, business logic layer, data layer etc., on each other, due to which modifications in some locations or interruption in operation affected other layers or the whole application. Moreover, there has been a growing need to be able to provide diverse services for heterogeneous technologies and bases, as well as for review of architecture so that it is compatible with further evolutions in iPhone or Android terminals, client instruments such as iPad, or high speed liquidity of general consumer market. At the same time, reduction in management costs was also being sought, and it was necessary to evade the increasing complexities in application logic management and system structures. As a design criteria to implement this, Rakuten explored different options for general simplification of the structure, by considering distribution into a multilayered system, and localization of effects brought on by mutual modification achieved by loose coupling of each layer.

2. Cost optimization: Performance improvement instead of hardware scale-up

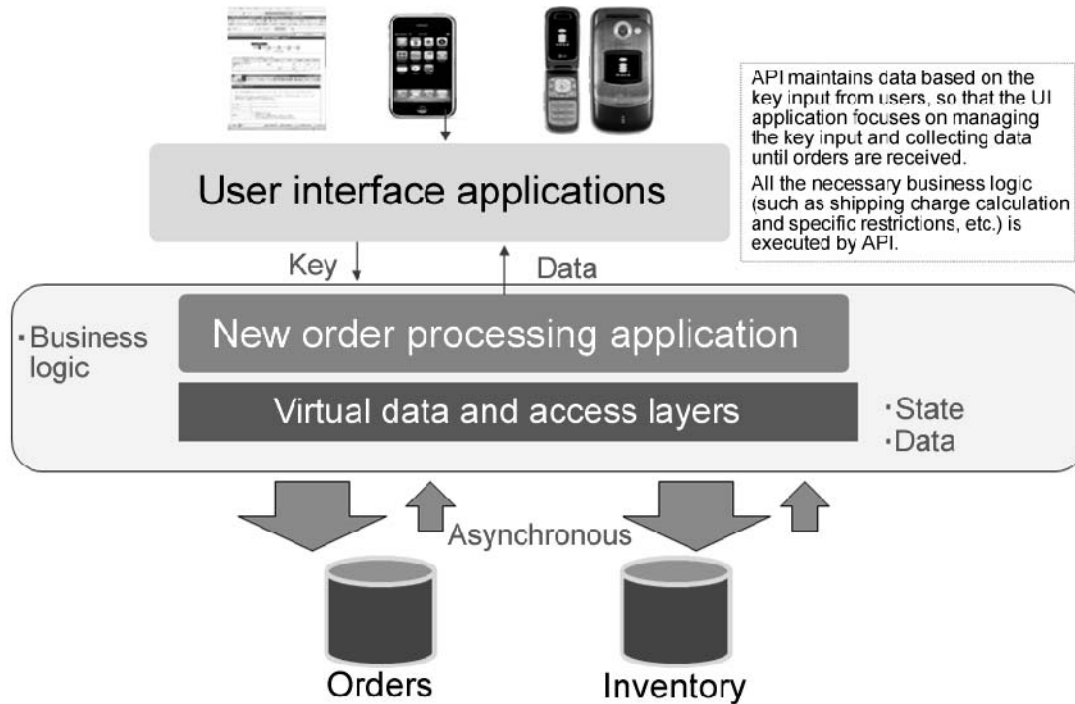
Rakuten considered a hardware scale-up for improving performance as a solution to remove DB bottleneck. However, since a performance amplification in future is inevitable, and assuming that scale-up would be a never-ending task, despite the fact that hardware becomes cheaper with further technological developments, its absolute value is extremely high. With sudden magnification of business scale, frequent and repeated scale ups far exceed the pace at which hardware becomes cheaper. Therefore, instead of a hardware scale up, it was necessary to look for a performance improvement method. One such approach is scale-out. There are products available in the market wherein performance amplification of data servers has been achieved by parallel use of multiple cheap version hardware such as blade server, etc. This is one of the options available to Rakuten for solving this problem. However, Rakuten opted for separation of application layer and data layer, by converting the entire system into a multilayered and loosely coupled structure, and explored more options that conform to this objective.

For implementing the aforementioned design criteria and concept, Rakuten focused on providing performance improvement together with a loosely-coupled, multilayered structure, and investigated the technologies to be selected.

Separation of UI layer and application logic (order process) layer or separation of application layer and data layer is normally implemented by inserting enterprise service bus (ESB) or message oriented middleware (MOM) between each layer. This approach uses a loosely-coupled simple system and is effective in resolving coordination issues between a great number of different applications. However, introduction of ESB complicates the base organization and management also becomes more complex. As per Rakuten, ESB is not always the best option for implementing a full width improvement in performance of large number of transactions or data processes while at the same time ensuring consistency of transactions with loosely-coupled simple structures. Although MOM is effective in construction of loosely-coupled structures, it was not considered to be the best option due to its complex base organization and management.

The company focused on DCP technology, that has asynchronous queuing or functions for distribution to many nodes and yet large quantity data and state can be maintained, and at the same time there is consistency in transactions. They acknowledged that this technology met all their implementation requirements. As a result, they selected DCP as a technology as a solution to the current problems and to implement the future goals. They adopted Oracle Coherence as the specific product (See Remark 3). Fig. 3 shows outline map of new applications that use DCP.

Fig. 3 Outline map of applications using DCP



Source: Created by Gartner based on Rakuten documents

While determining the future trends, Rakuten thought it was impractical to implement the designing tasks required for its adoption all at once, and decided to implement them in the following 3 phases (See Fig. 4 for overview of steps).

- Phase 1: Consideration of performance improvement (Jan ~ March 2010)

Their first priority was to use DCP as a simple cache to reduce load on DB load, so as to improve the performance. This task has been implemented.

- Phase 2: Separation of application layer and DB layer (August 2010 ~ present [on-going])

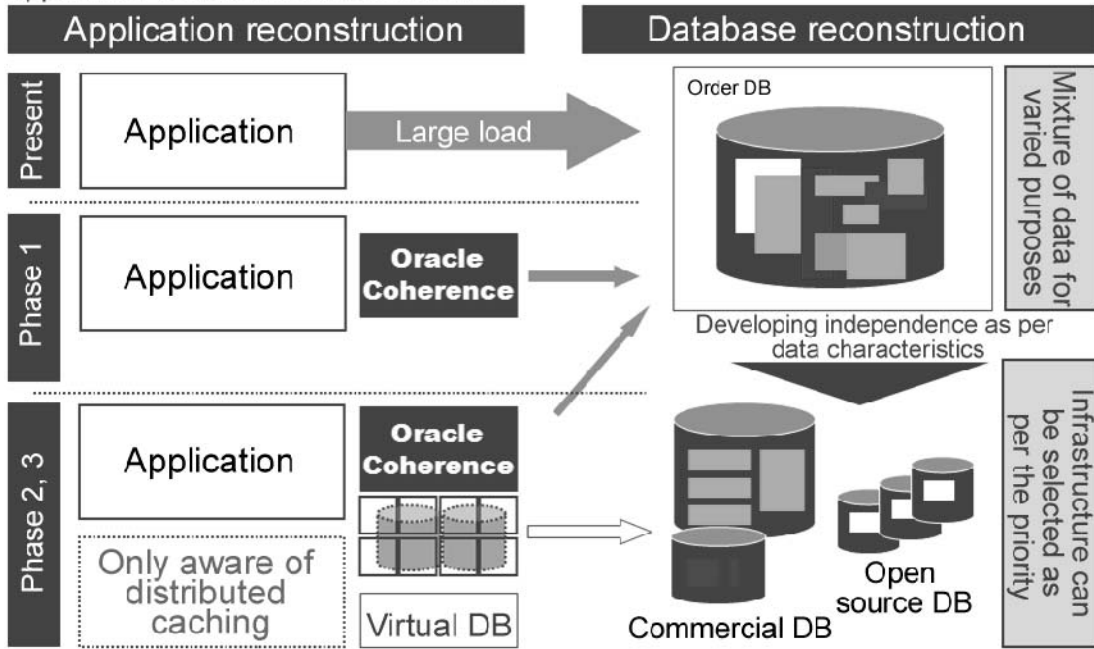
Completion of a structure wherein DB is hidden from the applications by deployment of DCP in front of the DB, and then abstraction of DB. With this they obtained a structure that does not let an application be aware of the type of DB being used, by embedding SQL in Coherence as a plug-in.

- Phase 3: Data model / data structure optimization

By using DCP, they aimed at localization of product-dependant parts or complexity of DB schema such as relational structures in data structure, so as to handle only the logical models in the application. Finally, they aimed at being able to select a DB product that is appropriate for data priority without having to reconfigure the application. They believed that quality of data management could be adequately ensured by repairing the Coherence plug-in and conducting a priority test.

Fig. 4 Steps taken by Rakuten towards a "Flexible structure"

Construction of a loosely-coupled layered structure wherein the DB and the application do not influence each other



Source: Rakuten

Results

Results and efficacy of the new application obtained with this method are as follows;

1. Performance improvement

DCP was introduced after converting the application structure into a loosely-coupled multilayered structure, to obtain a structure that accepts all data access requests from all sorts of applications (of both sides, consumer system and service operators). Results showed significant improvement in UI response time, to the extent that there was a margin for future increase in levels of services offered.

2. Results in terms of architecture

1) Successful separation of screen transition and business logic

Avoiding application statelessness and session control by using DCP functions

By completely entrusting the UI, application logic, and state management with DCP and state management application processing logic coupled with DCP, there was no further need to manage these processing states in individual UI applications. Even if the order processing application (application programming interface [API]) or parts of UI are down due to some reason, overall consistency is maintained by DCP, thereby obtaining a structure wherein the operations do not get impaired. Additionally, there is no further need to control requests from Web in Cookie and Web servers for distributing them into physical servers that maintain data, or to manage session data replication in the application server being done up till now, thereby making the system simpler.

Irrespective of where the web request was dumped, the use of DCP has made it possible to return the same outcome (See Remark 3).

Stateful order processing application when judging from UI application side

When judging from the Web UI application side, the integration of DCP and order processing applications was recognized as an application that manages the state. It means that if Web UI passes the same key to order processing application, always the same data is returned. (See Fig. 3). Consequently, with the adoption of DCP, statelessness peculiar to web applications and statefulness valid in transaction management could now coexist and a simple structure could be maintained.

This kind of a structural change brought about the following results.

- Reduction in development and production costs (although it differed as per the case, an increase in productivity up to 6 times was seen)
- Decline in threshold of skill sets related to UI
- Reduced dependence on DB skills (can be created just with the knowledge of UI application framework)
- Migration to a UI framework with higher productivity is possible

It became possible to develop UI logic and application logic separately.

Additionally, using the same service, it was now possible to implement optimal screen transitions in multiple client instruments. It is difficult to implement screen transitions similar to those done in PC in mobile terminals or smart phones since UI operability and information volume are different for different client instruments. However, by separation of UI and application logic, it was possible to implement screen transitions that match the device or utility by calling a common service logic.

2) Defining services with appropriate granularity

This efficiency in itself is a result of granular designing that makes SOA aware of the application design, rather than being a result of architecture or DCP introduction. It is defined as an operational level where the capabilities of each service can be understood at a glance. With this they were able to materialize a new integral application logic with integration of such services, or were promptly able to create a new UI. After activation of the new system, services for mobile telephones can be expanded rapidly, and this system contributes towards acquisition of business opportunities and expansion at a faster speed.

3) Liberation from limitations of operation interruption accompanied with DB maintenance

Separation of application logic and DB with loosely-coupled structure (currently under development)

State wherein purchase of product is settled was maintained using DCP, but process for reflecting this state in the DB was updated asynchronously. Owing to this, coordination between individual applications and DB, or tight-coupling between multiple DBs was no more required, nor was it required (loose coupling) for the application to be aware of the dependence on DB (data consistency, persistence).

Results showed Rakuten to be reaching one step closer to achieving their 100% targets. Even when operation interruption is required for maintaining the DB, all the data required for service operations is maintained in DCP, and therefore there is no need to interrupt the service itself.

Rakuten not only aims at separation of DB from the application but also at elimination of dependence on special DB products, by abstraction of DB itself. Coherence only considers put/get models as data access API, and as per Rakuten it was possible to allocate roles by accomplishing this objective, since SQL for DB access can be hidden as a Coherence plug-in.

4) Combinations of diverse functions (server side mashup)

Although functions required for purchase of products at Rakuten's website are provided by the order processing application, when parts of these functions (ex: inventory process) are to be used in a different application, a server side mashup at the UI side can be done by servicization of these functions.

In this way Rakuten implemented a loosely-coupled multilayered structure as an application architecture wherein even with a full width increase in number of users, there is no need to rely on hardware reinforcement, and overall system availability can be improved along with satisfactory maintenance of transaction performance.

Major Success Factors

Since Oracle Coherence, a DCP product, used this time does not use SQL language for data access, a large scale overwriting was required for migration of the existing application to the new environment. Results of overwriting to the application are already known, and motivation within the company is high as the prospects of tangible implementation look promising.

However, Rakuten adopted a stepwise approach of cautiously implementing application logic and data into Coherence instead of migrating the existing applications all at once. For example, while using the traditional logic (1st access from the DB), they used caching from the 2nd access onwards, and after confirming efficacy of restricting the application scope, they slowly expanded this range. This can be considered as the best practice for migration for application transformation.

Additionally, since DCP products have unique properties, hard coding of access to DCP interface in each application was not done and it was designed to stand between the interface access module as a buffering agent so that the degree of dependence on individuality of specific products does not increase, and it is possible to migrate to other products or technologies in future.

Future prospects

Aspiring the completion of Phase 3, Rakuten is advancing with SOA of each application and is making continued efforts for further improvements in flexibility of system structures based on the experiences gathered in phase 1.

Remark 1 Rakuten Inc.

Head office address: Shinagawa-ku, Tokyo

Established: February 1997

Capital: 107, 650 million yen (as of December 31, 2009)

No. of employees: Stand alone: 2,625, connected: 5,810 (as of December 31, 2009)

Description of business: E-Commerce, credit cards, banking, portal media, travel, securities, professional sports, communication

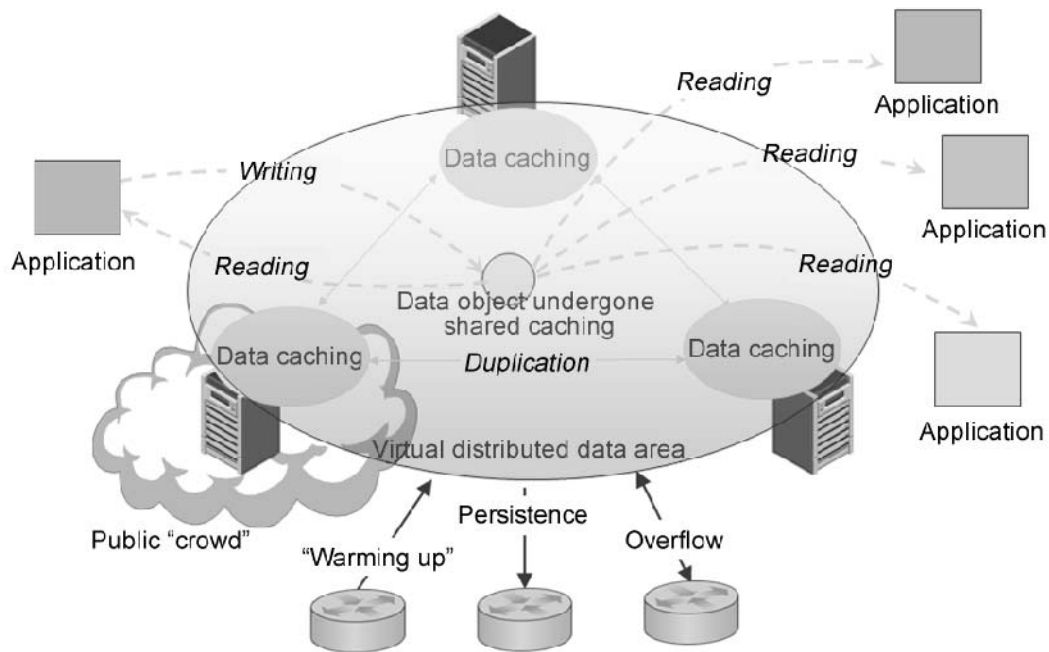
Remark 2 Distributed caching platform (DCP)

In DCP (depending on the vendors, also called “data fabric” or “information fabric”), important data is placed at a location (in memory) as near to the application as possible, with the objective of achieving a full width performance improvement as well as scalability of a data-consolidation-type application. In the application, operations can be performed by storing the data (possibly object) that has been asynchronously extracted via an adapter from various data sources such as relational DBMS (RDBMS), realtime data feeds, XML text, asynchronous messages, API, etc. at a storage location (cache) within the memory provided by this platform (See Fig. 5). Normally, objects in cache are operated through peculiar API or generic API (e.g., Java Database Connectivity [JDBC] or JMS) at the application program side, but some of the products can be transparently added to lower layers of the application by declaring the properties related to data structure that shares through cache.

DCP runtime roles include; initial loading to cache, extracted source data and cache state synchronization, cache object locking, transaction management, and cache event notification sending. Data maintainability during system fault can be ensured by distributing cache into multiple physical servers and further improving its performance as well as by replicating the important data. Hence, in addition to cache mirroring and partitioning, DCP is also equipped with clustering and fail over management and also supports the required security functions (for protecting the data from unauthorized access) and management functions. DCP can be added to the Java EE application server, .NET, or other platforms as plug-ins, and in these parts, objects in cache can be shared through cache distributed between multiple platforms (e.g., Java EE and .NET). Consequently, DCP can also be used as an ultra-high speed “publish / subscribe” engine with low latency. Moreover, it has products that support “stored procedure method” mechanism. With the use of such functions, data in cache can be processed by calling the application code residing in cache from the application.

Caching is a technique widely used in DBMS, Web server, application server, and other systems software. However from the application developer’s point of view, it is normal to have transparency in case of this kind of caching. Its use is limited to support of only special type of data (e.g., web pages or HTTP sessions) and the functions related to clustering, replication, partitioning, and data maintenance being prepared in DCP have not been implemented. DCP products are often compared with in-memory DBMS, however they do not provide complete support related to relational and semantic provided by Oracle’s TimesTen, IBM’s Solid, or other similar in-memory RDBMS.

Fig. 5 Application style in cloud age: Distributed transactions and distributed caching



Source: Gartner

Remark 3 Data consistency maintenance / transaction consistency maintenance in multiple Coherence nodes

For maintenance of unified data consistency in multiple Oracle Coherence nodes, data is copied in multiple nodes, due to which memory of multiple nodes is not shared by Coherence.

Fundamentally there is only 1 special data body in the shared memory area constructed on multiple servers, and at the time of receiving a data request, multiple servers maintain centralized data consistency on the shared memory by constantly accessing this 1 data body through the network.

While creating (writing data in coherence area) this single data body, a backup is simultaneously created on a separate server. This backup creation is a synchronous process and once data writing is complete, 1 backup body (1 by default) is always created.

In this way, by maintaining a single individual data body, there is no problem of inconsistency between servers in multiple Coherence areas (even if there is time lag in update access request between nodes) even if there are data update processing requests from multiple nodes.

Besides this, data storage area expansion and load distribution of data access processing can be achieved by distributed maintenance of this data group on multiple servers.