USING COOPERATION TO IMPROVE THE EXPERIENCE OF WEB SERVICES CONSUMERS

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the Degree of Master of Science

In the Department of Computer Science

University of Saskatchewan

Saskatoon

By

YUTING LUO

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan, S7N 5C9

ABSTRACT

Web Services (WS) are one of the most promising approaches for building loosely coupled systems. However, due to the heterogeneous and dynamic nature of the WS environment, ensuring good QoS is still non-trivial. While WS tend to scale better than tightly coupled systems, they introduce a larger communication overhead and are more susceptible to server/resource latency. Traditionally this problem has been addressed by relying on negotiated Service Level Agreement to ensure the required QoS, or the development of elaborate compensation handlers to minimize the impact of undesirable latency.

This research focuses on the use of cooperation between consumers and providers as an effective means of optimizing resource utilization and consumer experiences. It introduces a novel cooperative approach to implement the cooperation between consumers and providers.

ACKNOWLEDGMENTS

I would like to sincerely thank my supervisor Dr. Ralph Deters for his continuous support and encouragement during my study. Dr. Deters guided me all through these two years and provided me with inspiration and direction for my research.

I would like to thank my committee members: Dr. John Cooke, Dr. Julita Vassileva, and Dr. Anh Dinh for their valuable feedback and suggestions on my thesis.

Special thanks to Ms. Rajitha Bakthula for her cooperation in the IPod experiments. It was such a pleasant experience working with her.

I would also like to thank Ms. Jan Thompson, Graduate Correspondent at the department of Computer Science, who has been very helpful throughout my study here and very considerate. Additionally, I would like to thank all other professors, staff, and students for their support.

Finally, I would like to thank my family for their love and support all the time.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BPEL	Business Process Execution Language
CORBA	Common Object Request Broker Architecture
CVS	Comma-Separated Values
EC2	Amazon Elastic Compute Cloud
HTTP	Hypertext Transfer Protocol
IT	Information Technology
JSON	JavaScript Object Notation
LRU	Lease Recently Used
OS	Operating System
PDA	Personal Digital Assistant
QoS	Quality of Service
REST	Representational State Transfer
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service-Oriented Computing
TPC-W	Transaction Processing Performance Council – Web
TTL	Time-to-Live
UDDI	Universal Description, Discovery, and Integration
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WS	Web Services
WSDL	Web Services Definition Language
WSFL	Web Services Flow Language
XLANG	XML-based extension of Web Services Description Language
XML	Extensible Markup Language

CHAPTER 1
INTRODUCTION

The emergence of Web Services (WS) has helped promote Service-Oriented Computing (SOC), and it is now seen as the most promising technology for building loosely coupled systems [1]. WS based on SOAP or REST have begun to replace other forms of middleware (e.g. CORBA) as a means of exposing legacy applications. Key to the growth of WS are standardization, interoperability, and reusability, which together enable WS to be used as building blocks of services that can span organizations and computing platforms.

WS are considered as one of the most successful technologies to realize the Service-Oriented Architecture (SOA), which, first introduced by Gartner in 1996, is now the premier design principle for business applications. As defined by Papazoglou et al. [6], from a technology perspective, SOA is a logical way of designing a software system to provide services to either end-user applications or to other services distributed in a network via published and discoverable interfaces. In the SOC paradigm, a service is usually a business function implemented in software, wrapped with a formal documented interface that is well known to the targets [6].

In a typical SOA scenario, three kinds of participants are defined shown in Figure1-1.

- The Service Provider: software agents that provide and publish the services
- The Service Registry: an agency that makes available services from various providers discoverable by service clients
- The Service Client: software agents that can find and consume services

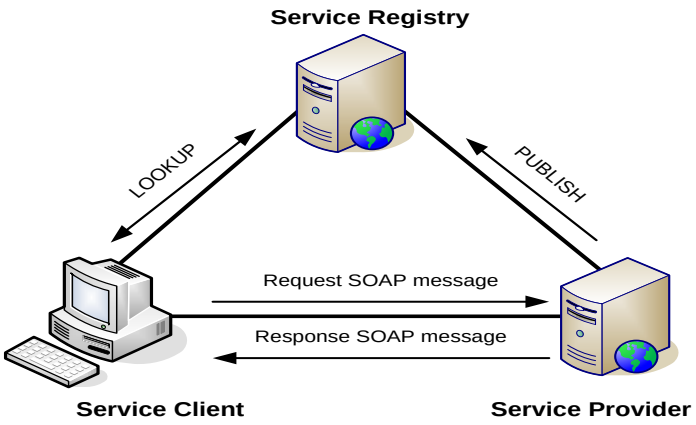


Figure 1-1. The Service-Oriented Architecture [25]

According to the concepts of SOA, the WS framework provides three mechanisms to help implement the SOA activities including publishing, discovering, and binding. The three mechanisms are the communication protocol, service description, and service discovery.

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:simpleReadA xmlns:ns2="http://calserverproxy/"/>
    </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:simpleReadAResponse xmlns:ns2="http://calserverproxy/">
            <return>3</return>
        </ns2:simpleReadAResponse>
    </S:Body>
</S:Envelope>
```

Figure 1-2. An example of SOAP Request and Response messages

The Simple Object Access Protocol (SOAP), initially created by Microsoft, is a lightweight XML based protocol for the exchange of information in a distributed environment. Standard WS often use SOAP as the communication protocol between the service provider and the service clients. Figure 1-2 shows an example of a SOAP request message and a SOAP response message. Rather than defining a new transport protocol, SOAP works on existing protocols such as HTTP.

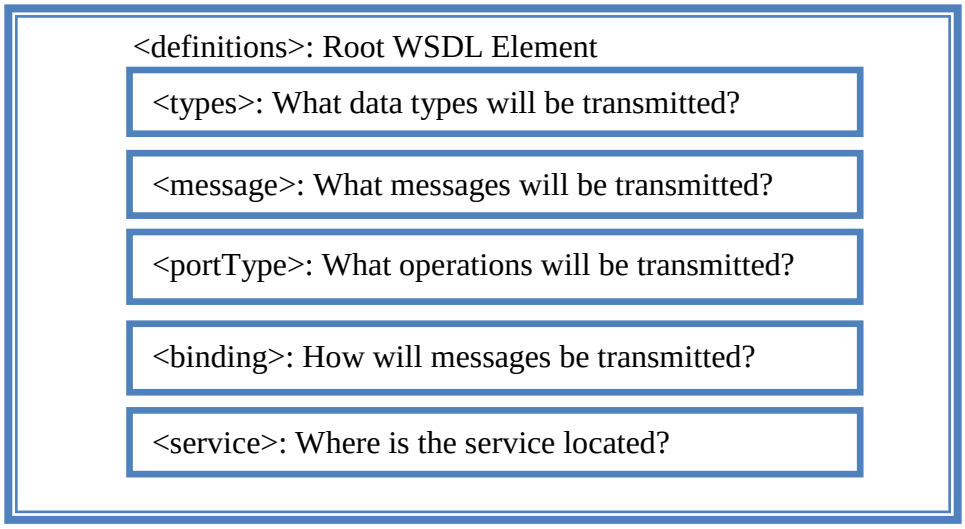


Figure 1-3. WSDL 1.1 specification in a nutshell [19]

For service description, WS employ the Web Services Description Language (WSDL) to describe the interfaces. The WSDL is an XML based format developed by IBM and Microsoft. A WSDL document is provided by the service provider. All the necessary information needed by the service client to bind to the service is included in the WSDL document. A complete service description provides two pieces of information: an application-level service interface description (abstract definition), and protocol biding details that service clients must follow to access the service at concrete service end points [16]. Figure 1-3 demonstrates the specification of a WSDL

document. The *types*, *message*, and *portType* elements are used for the abstract service interface description; the *binding* and *service* elements are used to describe the concrete binding information.

The Universal Description, Discovery, and Integration (UDDI) interface, which provides information regarding service categorization, service contact details, and technical data [16], is used by the Service Registry to list the available WS. The UDDI specifications offer users a unified and systematic way to find service providers through a centralized registry of services that is roughly equivalent to an automated online "phone directory" of WS [16].

More recently, RESTful WS began gaining popularity. For example, the Facebook API only uses the REST style interface. The acronym REST stands for Representational State Transfer, which is an architecture style for distributed systems introduced by Roy Fielding [17]. The central emphasis of REST is a unified interface for resources. In the REST style, services are viewed as resources that are identified by URIs and realized as representations. Clients consume services by accessing resources using standard HTTP commands such as GET, POST, PUT, and DELETE. As a result, read and write operations can be easily identified in REST and services using the Get command can be marked cacheable. The most popular formats used for a server response in REST are XML, JavaScript Object Notation (JSON), and comma-separated values (CVS). Compared to SOAP, REST is light weighted and simple. A request to access a REST service is just as easy as a HTTP command. While REST only uses four "verbs" to deliver all the services, SOAP based WS have much richer semantics in the operations.

The debate on SOAP and REST has not ended. Pautasso et al. [49] compared RESTful WS with SOAP based WS and concluded that REST is well suited for basic, ad hoc integration scenarios, SOAP is more flexible and addresses advanced Quality of Service (QoS). Whichever

style WS are using, they should all be published, found, and consumed through the web. As there are more and more services emerging and many of them are expected to deliver similar functionalities, the service providers compete for network and system resources such as bandwidth, as well as competing for service clients to achieve high business profits. Apart from the functional properties that most WS can deliver to service clients, non-functional aspects such as quality become important and are often a major concern for both the service providers and the service clients. QoS for WS is considered a key concept in distinguishing between competing WS [45]. QoS of Services for WS defines various quality requirements including performance, reliability, scalability, capacity, robustness, exception handling, accuracy, integrity, accessibility, availability, interoperability, security, and network-related QoS requirements [8]. However, how to provide a service with guaranteed QoS support remains an open question.

To ensure QoS, especially the performance aspect, various techniques have been studied such as caching and admission control. A key problem in current SOA research is that it assumes that service negotiation and service consumption are performed separately, which means the service provider and the service clients negotiate the QoS contract first and after that they start the service interaction. This is fine when penalties agreed in the QoS contract and compensations can fully handle the issues of QoS violation. However, this still results in the loss of time and resources which have already been spent on the unsatisfied or failed service interaction. Furthermore, consequences of the QoS violation in one node can spread to the chain of services as in the WS environment many business services are integrated [50]. Last but not least, this mechanism still cannot give a cure for unpredictable QoS problems that happen at run time.

This research investigates a new concept of cooperation in WS. Previous research on cooperation such as [55] mainly focuses on dynamic service selection for WS composition. In

this research, the cooperation is between the service provider and the service clients. It is mainly achieved by cooperative resource reservation, which is beneficial to both the service provider and the service clients in resource utilization. In order to achieve mutual understanding for better cooperation during the service interaction process, this research utilizes caching, prefetching and locking techniques to support cooperation between the service provider and the service clients. Experimental results are collected to evaluate the proposed cooperative approach.

The rest of this thesis is organized as follows: chapter two states the problem definition. Chapter three gives a literature review on QoS issues for WS, workflows, and locking. Chapter four presents the idea of the proposed cooperative approach and the design. Chapter five shows results of the experiments on basic performance parameters. Chapter six presents the evaluation with mobile devices. Chapter seven concludes this research and outlines promising future work.

CHAPTER 2
PROBLEM DEFINITION

The "*experience*" of service consumers has become an important factor in e-business, as well as in WS, in which the experience refers to the client's perception of service performance, service availability, etc. Improving the experience of WS consumers is of great importance to the service providers. Due to the heterogeneous and dynamic nature of the WS environment, to ensure good quality of services is non-trivial. While WS tend to scale better than tightly coupled systems, they introduce a larger communication overhead and are more susceptible to server/resource latency. Current approaches promote QoS support for WS.

In QoS enabled WS, the service provider publishes the QoS statement together with its service description document, indicating that it can provide service clients with services that have a quality as good as specified in the QoS statement. The service client then looks up services with QoS support that matches what it requires. If the lookup process is successful, a QoS negotiation process is thus completed, which means that both the service client and the service provider have agreed upon a QoS contract- Service Level Agreement (SLA). After that, both parties will only be responsible for either service provisioning or service consumption; very limited conversations on quality improvement between the service client and the service provider continue.

Challenges in QoS Support for Service Interaction

Although a QoS contract has been agreed on before the service interaction, the service clients do not know the actual quality of service (the service clients may know the average quality of service as stated in the contract) until they consume it at runtime. Moreover, both the service client and the service provider barely have enough knowledge of each other at runtime. As a matter of fact, the runtime environment is highly dynamic. It is usual for the service clients

to encounter a long period of time to wait for the responses. An even worse situation arises when the clients get no responses at all from the server. If no communication between the two parties takes place to solve problems caused by the unexpected changes, the client-perceived quality of service usually degrades.

One challenge in QoS support for WS is that costs are increased for both the providers and the clients to maintain high quality of service in the dynamic environment. The typical behavior of services under different loads is shown in Figure 2-1. As load increases, the performance of the server decreases. In order to keep competitive performance of the services, the service providers usually tend to increase the computing power to serve the peak load. For example, Google has tremendous computing power to serve its clients. However, the load is below the peak most of the time. Thus the computing power is underutilized and the costs increase. From the client's perspective, this is also the case since clients tend to pay more money to prepare for the worst situation, which consequently causes high costs.

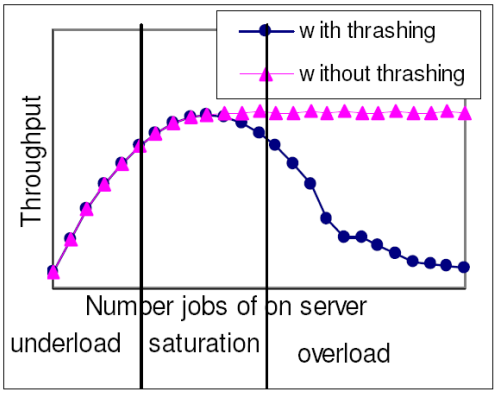


Figure 2-1. Service behavior under various loads [41]

Fortunately, cloud computing brings a solution to these problems. Cloud computing is the latest paradigm emerging to deliver IT services as computing utilities. It dynamically provisions

computing resources and services on demand as a personalized resource collection through virtualization technologies [54]. Companies can now rent computing infrastructure as well as software services at a good price. High fixed cost of hardware purchasing and maintaining, and software developing now can be avoided. Amazon Elastic Compute Cloud (EC2) provides resizable compute capacity with low prices based on hours. For example, the current rate for the most basic instance type is set at $0.10 per instance hour. Companies can easily expand their computing power to handle periodic traffic spikes by just purchasing more compute capacity from Amazon [53]. A prerequisite for this model to work efficiently is that the companies should know when their traffic spikes occur so that they can prepare ahead of time. Furthermore, if the companies can predict the time they need for the computing so that they can reserve necessary resources from the provider in advance, and the provider can coordinate the resources well before service interaction, thus both parties can benefit from this cooperation. The benefits include reduced costs, improved performance, and enhanced consumer experience.

Apart from those unexpected causes such as hardware failures, another factor that contributes to the dynamic is real time resource conflicts. Consider the following example as shown in Figure 2-2:

Client A sends a request to the service provider to access resource X. A plans to do a read operation on X for about 30 seconds. Almost at the same time, client B sends a request to the service provider to do a write operation on resource X. Then the conflict happens. Traditionally, B either needs to wait for 30 seconds to continue its request, or it has to cancel the request if it cannot afford to wait for such a long period. This approach to deal with the conflict is fair. However, more adaptable resolutions could be discovered for different scenarios.

Scenario 1: Client A and client B are unfamiliar with each other, thus they are opponents to each other; they compete for the resource in any case.

Scenario 2: Client A and client B are business partners, thus they are partial opponents to each other; they compete for the resource but results may be negotiable.

Scenario 3: Client A and client B belong to the same organization or entity, thus they work for the same purpose and goal; they should not compete for the resource.

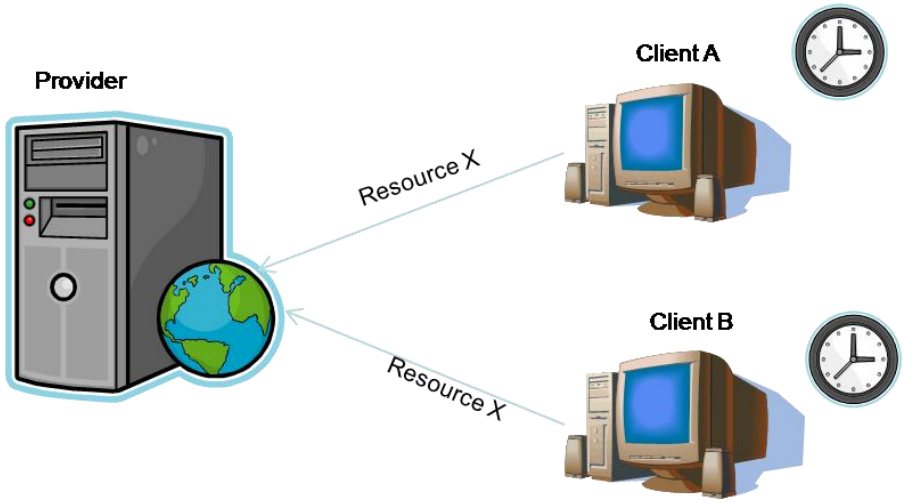


Figure 2-2. A scenario of resource conflict in service consumption

Here is an interesting and not-so-abstract example of scenario 3: a human user is driving a car with an automatic navigation device which gets services from a road traffic provider. The user also has a mobile device. While the car is navigated by the navigation device, the user is playing with the mobile device consuming services from the same road traffic provider. Now conflicts can happen. Since bandwidth is limited and the wireless network is not so stable, the human user of course should give the automatic navigation device the first priority to consume the services.

The Cooperative Behaviors

This research focuses on improving the experience of service consumers at runtime by introducing cooperation among service participants during the service interaction process in terms of service availability and responsiveness.

Generally, cooperation refers to working cooperatively to achieve the goal of a win-win situation. In this context, the concept of cooperation is made more concrete by using two tasks. One is exchanging information among service participants (this includes cooperation between the service consumers and the service provider as well as among service consumers); the other is re-negotiating at runtime. Cooperation between provider(s) and consumer(s) requires the ability for both to communicate and exchange messages. Depending on the scope of the cooperation (e.g. how many consumers and providers can cooperate), different cooperation scenarios are possible. Below are listed the three basic cooperative behaviors that have been the focus of this research.

Closed Consumer-Provider Cooperation

This is the basic cooperation scenario. The word "closed" in this situation means that the service clients and their information are not exposed to each other. In this scenario the provider engages N service clients but each client is not aware of the other service clients. The server provides a locking/reservation table and the client provides an augmented workflow. Since the clients are unaware of other clients, they are constrained to booking/reserving services and providing compensations in case of unexpected loss of service/resource. So in this scenario, the cooperation is only between the provider and the clients.

Open & Centralized Consumer-Provider Cooperation

This scenario extends the previous one by making service clients aware of the certain actions of some service clients. An augmented workflow and a friend list should be provided by the service client in order to complete the cooperation. A friend list is a list of business friends or partners of a client. It is used for cooperation among service clients. The provider offers access to the locks in form of a table. However, the clients can be informed of current owners of certain locks if conflicts happen. In this scenario potential lock conflicts can be solved by allowing lock swapping within a friend group. To determine if such a swap is permissible, each client is requested to rank the importance of its request and the server then determines the winner. This scenario is particularly interesting when multiple wireless devices (e.g. PDAs and mobile phones) compete for resources on behalf of a human user. Using friend lists and ranking requests is an approach for avoiding competition or even deadlocks among the devices of a single user.

Open & Decentralized Consumer-Provider Cooperation

By using the previous scenario but moving the locus of decision making from the provider towards the service clients is a more decentralized approach. Rather than relying on the server to determine the outcome of request competitions, the clients themselves evaluate not only how important their own requests are but also evaluate their opponents' requests. The evaluations are sent back to the server which averages them and selects the winner. So in this scenario, the cooperation is among all the participants of the service consumption process.

Since cooperation needs at least two parties to be involved, two proxies that represent the two parties are needed for the implementation of cooperation. One proxy named the Client-Proxy (CP) and the other named the Server-Proxy (SP). These two proxies are responsible for the communication between service clients and service providers at service consumption.

The Client-Proxy helps the service client to inform the service provider of its future requests. Prefetching is a technique that can be used for this purpose. Before the service client actually sends a request, the Client-Proxy should prefetch the client's next request(s) according to its prediction and then send the relevant information of the request(s) to the service provider. The client provides its workflow, which contains information about the flow of operations, to the Client-Proxy for prefetching. As a result, a client workflow model is needed for the prefetching. Additionally, caching techniques can be applied together with prefetching since it can improve the performance of the WS.

The Server-Proxy helps the service provider to inform the service clients of its status such as resource availability. Since the service provider may need to serve hundreds or even thousands of clients who request server resources for a certain period, it is necessary to have a computing component, which can coordinate the resources and have the global knowledge of the status of each resource, in order to inform the service clients correctly. This job can be completed by the Server-Proxy. Ideally, the Server-Proxy should have the ability to coordinate resources associated with time periods since service clients may request resources at different times. In this research a reservation based locking protocol for the resource coordination is proposed. The Server-Proxy should also have the ability to initiate a re-negotiation with the Client-Proxies in case of changes such as resource unavailable at runtime. Policies of different negotiation strategies can be defined.

In conclusion, the key point of the success of cooperation is to share enough information with each other. As long as there is enough information, conflicts can be easily dealt with. This leads to the following research questions:

- What information is needed to be shared?
- When to share the information?

- To whom the information is open?
- How to use the information?

This research will explore those questions as well as investigate the costs and benefits of different cooperation approaches to support QoS requirements for WS.

CHAPTER 3
LITERATURE REVIEW

This chapter reviews QoS in WS, approaches to improve the performance aspect of QoS in WS, Locking, and Workflows.

QoS in WS

Quality of service (QoS) is a general term for a set of technologies and mechanisms which allow applications to consume services in a guaranteed manner [2]. Due to the heterogeneous and dynamic nature of WS, they pose many new challenges introduced by increased latency and trust issues as services can span multiple organization or systems [43]. To overcome the new challenges, current research focuses on providing QoS support, since guaranteed QoS brings competitive advantages for service providers and supports a good "*experience"* for the service consumers. QoS is being studied and deployed in many ways, such as defining and modeling new QoS metrics, integrating policies, and introducing middleware [5] [43] [52] for non-functional aspects. As stated in the World Wide Web Consortium (W3C) standards [8], a variety of QoS requirements have been identified, such as performance, reliability, availability, and security, etc. Some of the major quality aspects are explained as follows.

- Performance: The performance of WS represents how fast a service request can be completed in terms of throughput, response time, latency etc. The service is considered well performing when it has a high throughput, low latency, and short response time.
- Integrity: Integrity is the quality aspect of how WS maintain the correctness of the interaction in respect to the source. Proper execution of WS transactions will provide the correctness of interaction.
- Reliability: Reliability represents the ability of WS to perform and maintain the agreed service quality.
- Availability: Availability is the quality aspect of whether the services are present or ready for immediate use.

- Security: Security for WS means providing authentication, authorization, confidentiality, traceability/audit- ability, data encryption, and non-repudiation.
- Price: the cost of WS usage with associated quality.

The above attributes mentioned are only a few among many possible QoS factors. Some researchers are focusing on indentifying more QoS attributes for selecting good WS. Maximilien et al. proposed a reputation and endorsement approach to select WS providers [21]. They modeled reputation in terms of sub attributes reflecting a user's experiences with a given service. Kalepu et al. proposed verity as another attribute in the QoS metric [20]. The verity of a service is measured by external components and is defined as the ability to maintain the lowest difference between the projected and achieved levels of service metrics.

For the standardization of QoS specification for WS, IBM proposed the Web Service Level Agreement (WSLA) framework in 2002 [22]. This framework aims at translating an SLA into configuration information for the individual service provider components and third party services to perform the measurement and supervision activities. The WSLA language specification is thus proposed for a detailed definition of the QoS parameters including how basic metrics are to be measured in systems and how they are aggregated into composite metrics.

Performance Aspect of QoS in WS

Among various aspects of QoS, the key factors that have a direct impact on the experience of service consumption are service availability and service performance such as responsiveness, as Erradi, Verma, and Olshefski discussed in their works [2][3][4]. This is because these two aspects are what the service consumers can perceive directly through service consumption. Monaco et al. [51] named them "the user-perceivable quality of service". Hence these two aspects are the most studied in QoS to ensure and support the success of WS.

The performance of WS is measured in terms of throughput, latency, and execution time. Throughput represents the number of WS requests served in a given time period. Latency is the round-trip time between sending a request and receiving the response. It can also be called the client-perceived response time, which is considered the most important performance parameter from the client's perspective. Execution time is the time taken by the WS to process a sequence of activities. Higher throughput, lower latency, and shorter execution time represent well performing WS. To improve responsiveness, many techniques have been studied. SOAP messaging is one area that attracts researchers on the messaging layer. Caching and prefetching are two other techniques that are widely used to reduce latency. Admission control has been broadly studied to balance load on servers as well as to maintain peak throughput.

SOAP/XML Messaging

The overall performance of the WS depends on application logic, network, and most importantly on underlying messaging and transport protocols such as the SOAP and HTTP it uses [23]. "The SOAP request begins with the business logic of the application learning the method and parameter to call from a WSDL document. This whole process is time consuming; it requires various levels of XML parsing and XML validation and hence hits the performance of the Web service", Sumra et al. explained in [23].

Mani and Nagarajan [24] pointed out that SOAP is the de facto wire protocol for WS. SOAP performance is often degraded due to the following:

- Extracting the SOAP envelope from the SOAP packet is time-expensive.
- Parsing the contained XML information in the SOAP envelope using a XML parser is time-expensive.
- Limited possibility of optimization with the XML data.
- SOAP encoding rules make it mandatory to include typing information in all the SOAP messages sent and received.

- Encoding binary data in a form acceptable to XML results in overhead of additional bytes added as a result of the encoding as well as processor overhead performing the encoding/decoding.
- The XML processor must be loaded, instantiated, and fed with the XML data. Then the method call argument information must be discovered. This involves a lot of overhead as XML processors grow to support more XML features.

As a result, many researchers focus on the message delivery part of the WS provision and consumption process and try to optimize the XML processing to reduce service latency since SOAP messages require extensive processing due to their representation. Abu-Ghazaleh [9] and Suzumura [10] proposed approaches for optimization of serialization of SOAP messages on the sender-side and deserialization on the receiver-side respectively to improve the performance of QoS in WS, since they consider the serialization and deserialization are costly processes. Serialization is a process of converting application objects passed from application logic to XML messages; and deserialization is the process of converting XML messages to application objects passed to application logic. Similarly, some research is dedicated to work on SOAP messages themselves for high performance, such as Chiu in his work [12] recommended SOAP extensions and a multiprotocol approach that uses SOAP to negotiate faster binary protocols between messaging participants. In Werner's work [13], differential encoding is introduced for SOAP message compression for higher application performance. Rosu proposed A-SOAP [44], which combines the optimization of SOAP message composition, message parsing, and message compression to reduce the SOAP related processing and communication overheads for WS.

Caching

Caching technologies are an effective way to reduce latency, as well as network traffic [25]. In WS, caching is applied broadly for the improvement of performance.

As the SOAP protocol is a bottleneck of the WS performance, Devaram et al. [18] proposed a client-side caching strategy to optimize the client SOAP requests. The experimental

results in their research demonstrate that the performance with respect to round-trip response time increases around 800%. Liu et al. [26] proposed a dual caching approach for mobile devices to achieve better service performance and service availability by caching both SOAP requests and responses on both the client side and the provider side using two proxies.

Friedman [27] proposed to cache WS in mobile wireless ad-hoc networks as a way of making such services more accessible to mobile devices. He mentioned that caching of WS should be organized as a service itself. Moreover, the caching service must be able to cache both the data and code that manipulates the data in order to be generic. Yin et al. presented a cooperative caching approach in ad-hoc networks in [47]. This cooperative caching approach involves mobile nodes to cooperate with each other by providing cached data and paths.

Takase et al. [28] described a response cache mechanism for WS client. They proposed three optimization methods to investigate the improvement of the performance of the proposed response cache. The first optimization is caching the post-parsing representation instead of the XML message itself. The second is caching application objects. The third optimization is for read-only objects. These methods reduce the overhead of XML processing or object copying. They showed through experimental results that these methods have large differences in their performance and various limitations; it is important to combine these methods properly in order to develop a high performance cache.

Prefetching

Although caching is an efficient way to improve web performance, it still has limitations. Wang [29] pointed out that regardless of the caching schemes in use, over half of the documents cannot be found in the cache due to the fact that the maximum cache hit ratio is usually no more than 40 to 50 percent; and one way to further enhance the performance of caching is to anticipate

future documents requests and prefetch these documents in a local cache. Wang categorized three patterns for prefetching in the web context:

- Between browser clients and Web servers.
- Between proxies and Web servers.
- Between browser clients and proxies.

Since prefetching relies on the anticipation of future requests, the efficiency of this technique is determined by the prediction algorithms [46]. Most of the existing web prefetching uses the dependency graph which presents the probability of future accesses according to the history access logs. Pallis et al. [42] proposed a clustering-based prefetching scheme for web prefetching. Their prediction algorithm is again based on the access log file; but the difference is that they cluster web pages for different clients according to the domains. Liu et al. [26] utilized a workflow based prefetching technique to enhance caching performance for mobile devices to consume WS. In their approach, two prefetching components are introduced. One component on the client side prefetches the client's requests based on a BPEL workflow file; the other component on the server side is responsible to prefetch service response messages. The two prefetching components follow the second and third prefetching patterns according to Wang's [29] categorization.

Admission Control

Admission control is another way of reducing latency. Admission control of requests is used to prevent systems from being overloaded and by enabling QoS guarantees in terms of response time and service availability [14]. Conventional works use a tail-dropping strategy to admit requests. This only works well in steady workload situations. More recent research on admission control shows other approaches to deal with dynamic situations. Elnikety's approach [14] controls admission of service requests based on estimates of request execution time and

server capacity. Requests that do not exceed the capacity will be admitted for processing; otherwise it will be deferred to execute later. Elnikety's work also proposes an aging mechanism, where an upper bound that a request is delayed in the waiting queue is defined, to prevent long jobs from starving in request scheduling. Although short jobs may have privileges to be serviced first, they can only be promoted to the front of a waiting queue if the promotion does not cause any pending request to be delayed more than its upper bound. Verma and Ghosal [15] presented a short term prediction based admission control that accepts requests which can yield maximum profits for service providers.

Locking

Locking is widely used in database systems to control data concurrency. The basic idea is, as C.J. Date noted [30], when a transaction needs an assurance on some object (e.g. a database tuple) it is interested in, it acquires a lock on that object so that other transactions will not disturb its execution. Basically a database system supports two kinds of locks, exclusive locks (X locks) and shared locks (S locks). X and S locks are also called write locks and read locks, respectively. The fundamental rules of locking are defined as follows:

- If transaction A holds an exclusive (X) lock on a lockable object (a tuple t for example), then a request from some distinct transaction B for a lock of either type on t will be denied.
- If transaction A holds a shared (S) lock on tuple t, then:
- A request from some distinct transaction B for an X lock on t will be denied.
- A request from some distinct transaction B for an S lock on t will be granted.

The Intent locking protocol is introduced to deal with the dilemma of finer locking granularity for grater concurrency and the cost of locking. Three types of intent locks are proposed: Intent shared locks (IS), intent exclusive (IX) locks, and shared intent exclusive (SIX) locks. The basic idea of intent locking protocol is that before a transaction can acquire a lock of

any kind on some object, it must first acquire an appropriate intent lock on the "parent" of that object.

Locking has been mostly discussed and implemented in databases for transactions. Until recently, the concept of locking has been applied to broader areas, such as the application level. Mock et al. [31] proposed a cooperative locking approach for cooperation in a distributed environment. Besides the basic S and X locks, they introduced a new type of lock named cooperative locks. Actions that hold cooperative locks on the same object are allowed to proceed but only in a hierarchical manner in which a later action is considered the sub-action of a previous one. Other actions without cooperative locks are denied to access the object. Zhao et al. [32] proposed a two-step reservation-based coordination protocol for WS transactions. The first step is an exclusive blocking reservation for a resource, and the next step is confirmation or cancellation of the reservation. In this protocol, the application has full control over the reservation activity. This means, differently from what conventional approaches did, the locking of a resource is no longer internal to the database system but controlled in the application level. For example, the application can decide how long the resources should be locked. However Alonoso et al. [48] pointed out that a central transaction coordinator who controls the locking of resources is needed to be redesigned to work in a fully distributed fashion and must be extended to allow more flexibility in terms of locking resources.

Workflows

A workflow is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules [34]. Workflows are designed for concealing the implementation details of the application, and for better understanding of the business logic [35]. A process definition should be provided to describe the business process. Typically, the process definition

contains details about the business process, such as the sequences of activities, rules for navigating between activities, conditions for starting and completion, participants to complete activities, and other related data.

To represent a workflow model, many languages and specifications have been proposed. The FlowMark model was introduced in 1994 by IBM to describe business processes. It defines several elements that are included in a workflow model [33].

- Process: a description of the sequence of steps to be completed. It consists of activities and relevant data. And it can be nested.
- Activity: each step within a process. It has a name, a type, pre- and post-conditions. Each activity has an input data container and an output data container.
- Flow of Control: the order in which activities are executed.
- Input Container: a set of typed variables and structures which are used as input to the invoked application.
- Output Container: a set of typed variables and structures in which the output of the invoked application is stored.
- Flow of Data: a series of mapping between output data containers and input data containers to allow activities to exchange information.
- Conditions: specify the circumstances under which certain events will happen.

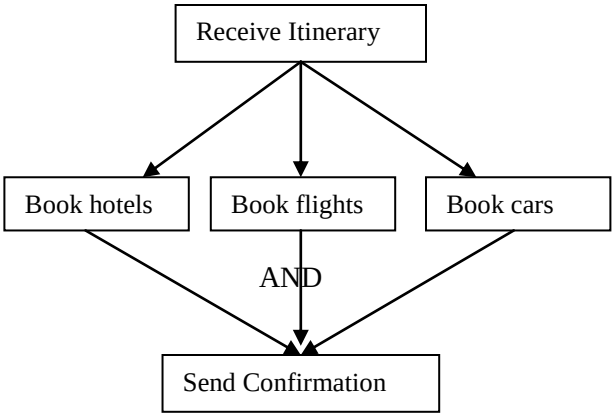


Figure 3-1. An example of a FlowMark workflow

Figure 3-1 shows an example of a reservation workflow represented by the FlowMark model.

The Business process execution language (BPEL) is a language that is proposed to integrate loosely coupled services into a business process workflow. It converges two early workflow languages: the Web Services Flow Language (WSFL) and the XML-based extension of Web Services Description Language (XLANG) [36]. Thus it combines both approaches and provides a rich vocabulary for description of business processes. Figure 3-2 shows an example of a BPEL process.

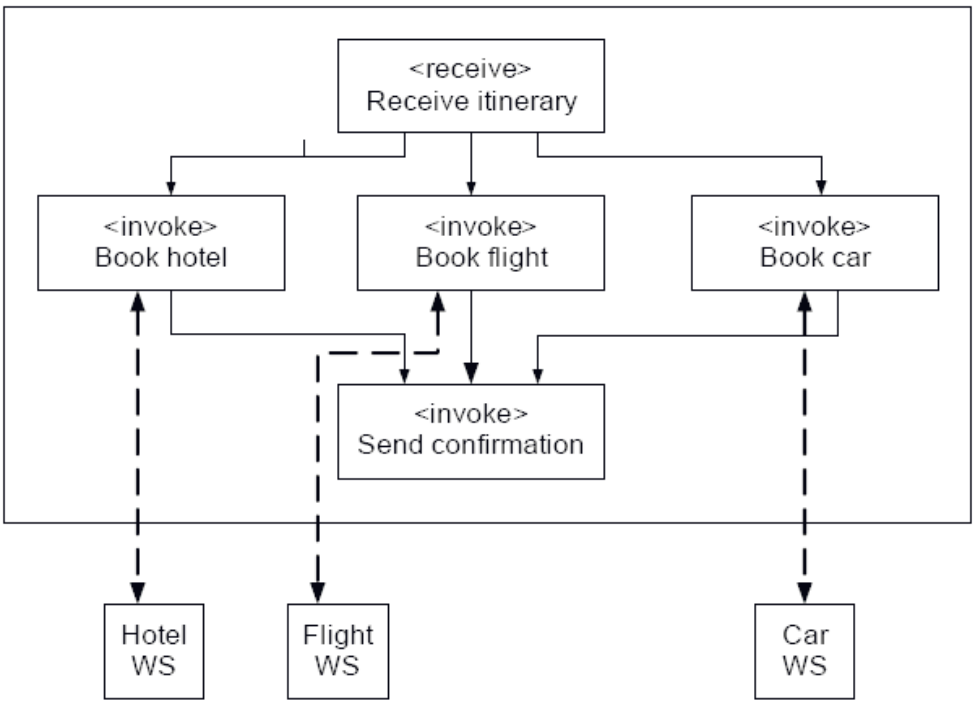


Figure 3-2. An example of a BPEL workflow [36]

The state machine has been in use since the advent of computing for a wide range of purposes. Recently, Microsoft introduced state machine style workflows in Windows Workflow

Foundation [37]. The reason that they employ a state machine to model workflows is that the state machine can provide an event-driven process execution with high flexibility. Figure 3-3 shows an example of a state machine workflow.

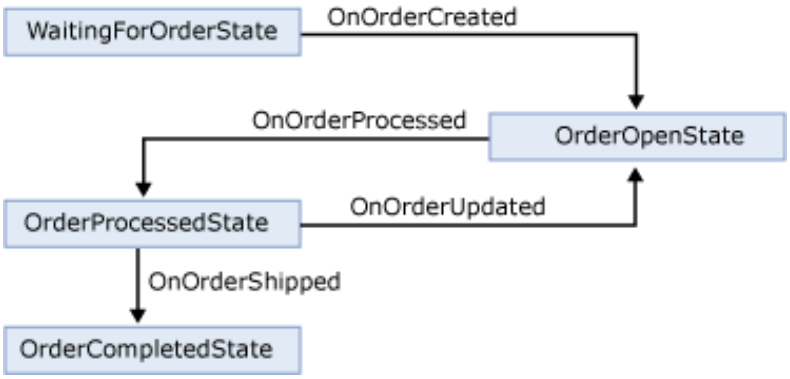


Figure 3-3. An example of a state machine workflow [38]

Summary

Previous work in QoS for WS provides a foundation for research on ensuring quality of WS with respect to performance which is an important issue in WS, especially from the client's perspective. Researchers working on SOAP messaging assume that SOAP is the de facto wire protocol for WS [24] thus software should comply with the SOAP standard. Caching and prefetching can efficiently improve performance by eliminating the transmission time under the assumption that read operations dominate. Admission control is conducted on two basic assumptions: the load that a particular job will generate is known and the capacity of the system is known [14]. Although research in this area has contributed a lot to improve performance for consuming WS, the runtime dynamic in the unpredictable environment remains an obstacle that restricts the progress. Moreover, very limited research focuses on dealing with runtime dynamics to improve performance of WS. Therefore, one of the goals in this research is to identify if the

proposed cooperative approach can be used to deal with runtime dynamics with respect to resource conflicts on the server side. Another limitation that lies in the previous approaches is that the service client is only considered as a pure service receiver and the server is the locus of control. This neglects the service client's capability of being cooperative, wasting the service client's potential in the effort of ensuring QoS in WS consumption. Hence, this research involves service clients as active participants in the process of service provisioning and consumption, aiming to enhance the service client's experience of service consumption in terms of various QoS aspects. Workflow based prefetching technique is employed for the implementation of client side cooperation. Table 3-1 gives a short summary of performance improvement techniques discussed in this chapter.

Table 3-1. Summary of performance improvement techniques

Techniques	Relevant works in this area	Common limitations
SOAP\XML Messaging	− Parsing of SOAP messages − SOAP messages compression	− Server (provider) is always the locus of control − Lack of runtime communication between clients and the provider − Clients are neglected for any contribution − Cooperation is seldom used
Caching	− Cache SOAP requests in client side − Cache both SOAP requests and responses using proxies − Cache Web services (the data and code) − Cooperative Cache	
Prefetching	− analyzing access log files for prediction − Workflow based prefetching using BPEL file	
Admission Control	− Admit requests based on estimation of server capacity and execution time − Admit requests that can yield maximum profits	

Locking is a traditional technique used in databases for data concurrency control; however, it has seldom been applied to runtime resource coordination in WS. In most cases, locking is completed at the database level; the application has little control over it. The open question is if locking can be used at the application level for resource coordination in the

unpredictable environment where WS are running. This research aims to identify the possibility of applying locking as a way to coordinate resources at runtime.

The workflow technology is often used internally for the automation of business processes in order to improve the business efficiency. In this research, a workflow file is also considered as a piece of cooperative information which is shared among the service clients and the service providers. The service clients can now use the workflow files to express what they need for service consumption.

CHAPTER 4
THE COOPERATIVE APPROACH

The cooperative approach aims to provide service clients with a good "*experience"* in consuming WS in terms of performance and service availability. Cooperation in this research refers to information exchange and re-negotiation between service clients and service providers at runtime. In this cooperative scenario, service clients are no more silent service recipients, but active information providers who express to the WS provider what services they need in the coming future, thus giving the WS provider enough time to get prepared for the future services. Accordingly, the WS provider should offer service clients easy access to resources, as well as to inform them about any change of services, thus leading to a re-negotiation process rather than only sending an error message or arbitrarily shutting down the services. A hypothesis of this research is: with enough information shared and exchanged, service performance can be improved under certain situations. Experimental results will be presented in the next chapter to test the hypothesis.

As stated above, the main idea is information sharing and exchange, but how to use the information and what exactly is the information?

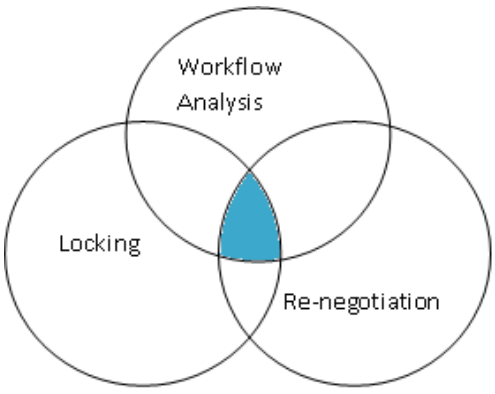


Figure 4-1. Techniques used in the cooperative approach

The cooperative approach mainly uses the three techniques shown in Figure 4-1. They are workflow analysis, locking, and re-negotiation. The intersection of the three techniques is the key to cooperation, since the cooperative approach combines the three techniques to implement the information exchange and sharing. The clients' workflow files will be shared and analyzed. So the service provider can coordinate resources using the locking technique and information retrieved from the workflow files. In case of resource conflicts, a re-negotiation process will be initiated by the service provider.

Architecture

In the cooperative approach, two proxies are introduced to implement the cooperation. One proxy works on behalf of the service provider, called Server-Proxy; the other represents the client, called Client-Proxy. Both of the two proxies appear transparent to service clients. The two-proxy architecture is enlightened by Liu's research on dual caching [25]. Figure 4-2 below shows the architecture.

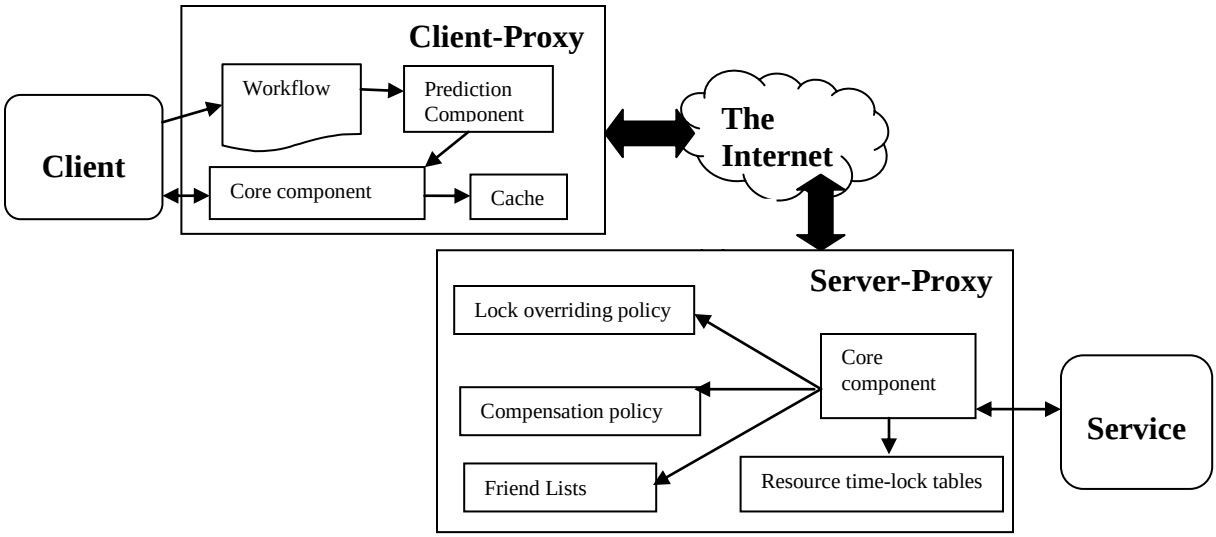


Figure 4-2. Basic architecture of the cooperative approach

The Client-Proxy is a small computing component that is hosted in the client's application. It communicates directly with the client and receives WS requests from the client, while at the same time it hides itself by providing the same service interface as the actual WS provider does. As a result, the service client is interacting with the Client-Proxy as it were interacting with the actual WS.

More importantly, the Client-Proxy is responsible for the implementation of the client's cooperation with the WS provider. This cooperation involves sharing part of the client's workflow with the WS provider, informing the WS provider what services it might consume and when to consume them. When the client's application starts, the Client-Proxy will prefetch the next WS request predicted by a simple prediction model (described in the next section) according to the workflow file provided by the client. Once a WS request is predicted, the Client-Proxy will inform the server of this request so that the server can arrange necessary resources for the call.

Another function the Client-Proxy provides is caching. Since the results of read operations can be easily cached, the Client-Proxy caches the response content of read operations. As a result, the next time the client invokes the WS call, it can enjoy better responsiveness. An invalidation thread will check the validity of cached items by sending requests to the service provider at a fixed time period or according to the TTL parameter.

The Server-Proxy is a component that releases the actual WS server from heavy resource coordination. It manages all the incoming WS requests and conducts resource coordination. The resource coordination utilizes a reservation based locking mechanism (see details in the locking section) for reserving resources. This resource coordination process is conducted prior to the actual service interaction. Both the Server-Proxy and the Client-Proxy are involved to complete this process. Once conflicts are detected in resource coordination, the Server-Proxy will initiate a

runtime negotiation process with the service clients to make a decision for resolving the conflicts. After a resource is successfully booked, the Server-Proxy will forward WS requests to the actual WS server for processing. The Server-Proxy can be implemented in different ways, from a simplest object living in the server application to an independent server either local or distributed.

A Simple Prediction Model for Prefetching

The workflow based prefetching technique can not only be used for web page prefetching, but also in WS [26]. Since every participant has its own business logic, a workflow can best represent the business process. And thus it can easily be integrated to the prefetching technique. In the cooperative approach, the service client will share part of its workflow with the service provider as a part of the cooperation. The Client-Proxy analyzes the workflow file to predict the service client's future interaction with the provider. And necessary resources are then prefetched from the Server-Proxy.

The prediction takes place when the client just finishes a service request; in other words, it happens at the time when the client just reaches a certain state. In the simple prediction model, the next service request is predicted based on both the current state and previous actions that the client has taken. This is because different sequences of previous states may result in different next states. Here, for simplicity the length of the sequence is two. As a result, three basic sets are defined:

S_p: the previous states the client visited;

S_c: the current states the client is visiting;

S_n: the next states the client will visit.

States are obtained from the workflow file. Particularly, some states have different identities at different times. For example, a state can be the next state when it will be visited

soon; it can also be the current state when the client is visiting it; it can be the previous state as well if it's just visited. To better organize the states and make use of them, they are gathered into groups according to the workflow. Every instantly related three sets of states: S_p, S_c, and S_n will be considered as in one group. Each group is called a path. Consequently, there are many possible paths. All of the paths should be derived directly from the workflow file so that they are all related. A full path that begins from the initial state and ends at the final state thus can be constituted by the atom paths.

A path can be represented by a tuple. In addition to the three sets which were talked above, a fourth set in the path tuple is introduced. It is the number of the path visits, denoted by V.

Path: (S_p, S_c, S_n, V)

The attribute V is used to predict the next operation when there is a choice in the workflow. Figure 4-3 gives an example of this situation. When the client's workflow enters state A_1, it then has two ways to continue. If condition C_1 is satisfied, B_1 will be the next state. Otherwise B_2 will be the next state. According to our model, this example can be interpreted by two paths: Path ({A_0}, {A_1}, {B_1}, {V_1}) and Path ({A_0}, {A_1}, {B_2}, {V_2}). Simply, the path with the largest value of V among all the possible paths will be chosen as the prediction result.

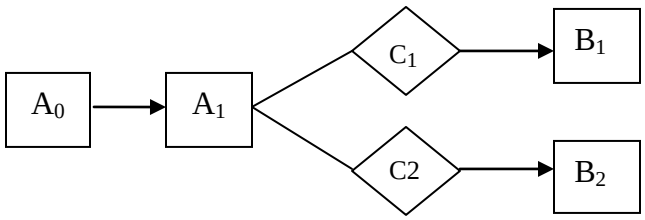


Figure 4-3. Choice Structure in a workflow

Other structures can also be described using this simple model. Figure 4-4 gives the details of the description.

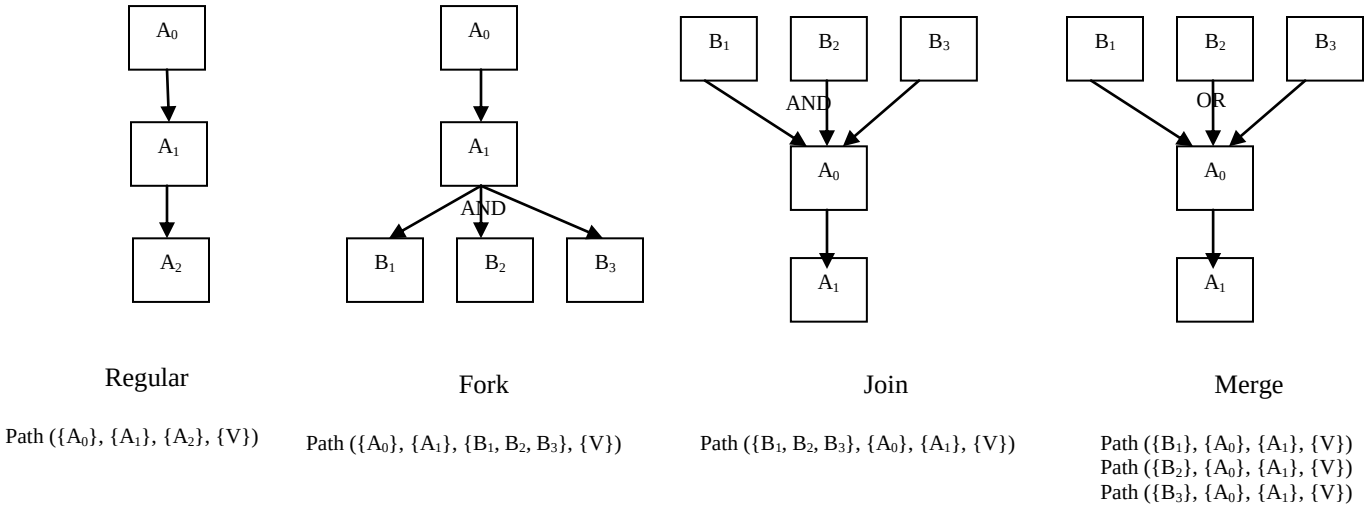


Figure 4-4. Node structures

The prediction algorithm is straightforward. Initially, all paths are obtained from the client's workflow and rewritten into an XML file. The Client-Proxy records the previous state and current state of the client, and looks for the right path(s) that match(es) the previous states and the current states. If more than one path is matched, the Client-Proxy selects the path with the largest V value; then the next operation is thus predicted as indicated in the path. Initially, every V attribute is set at 0. Once the client's application starts, the Client-Proxy also starts to work. Every time the client invokes an operation, the Client-Proxy will be notified of the event and 1 will be added to the attribute of V that belongs to the path where the client is walking through. And the next operation is prefetched by the Client-Proxy from the paths file as predicted. Then the server will be notified of the operation. Figure 4-5 shows the algorithm.

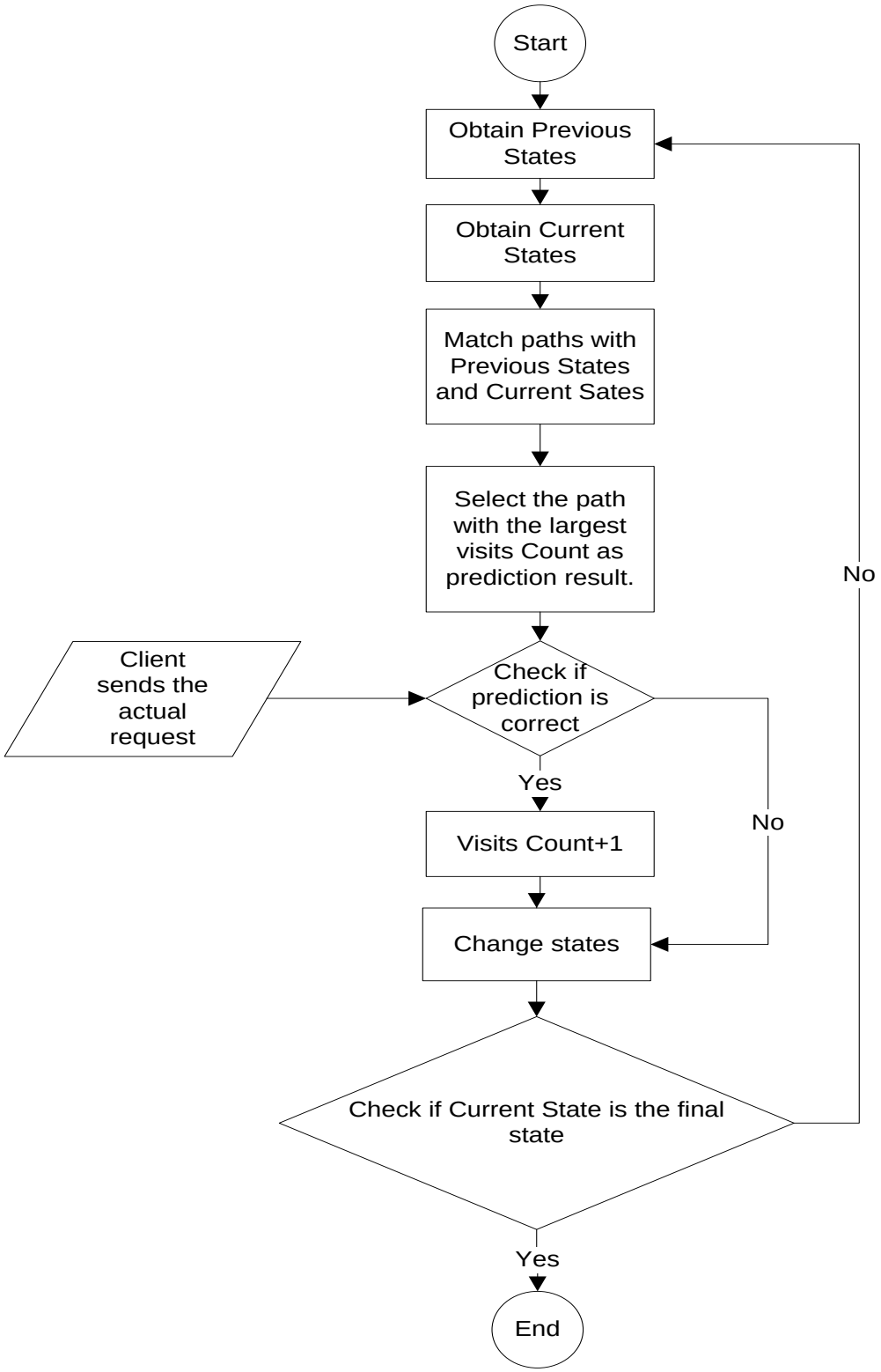


Figure 4-5. The prediction algorithm

A Reservation Based Resource Locking Protocol

As indicated above, after the client performs an operation, the Client-Proxy, representing the client, will inform the server of the next operation that the client might perform. And the service provider can use the information from the Client-Proxy to coordinate resources for the future operation as a response to the client's cooperation. To inform the provider, pre-requests are used; and for the coordination of resources on the provider side, a reservation style resource locking protocol is proposed.

Pre-requests

Pre-requests are requests generated by the Client-Proxy prior to the actual WS requests, and right after the Client-Proxy's prediction of the next operation. They are used for informing the WS provider about the client's possible service consumption. Additionally, they are used for reserving server resources (e.g. a token, a lock) for the client's actual WS requests. As a result, a pre-request should contain information about both the client and the operation. The following parameters can be defined in a pre-request:

prerequest (client,priority,service,time,dur,lock,token)

client: the client ID (should be unique)

priority: the client's priority

service: the name of the service

time: start time of the service

dur: duration of the service

lock: the operation lock required for consuming the service (described in the next section)

token: the token already held for booking the lock (described in the next section)

The Server-Proxy is responsible for handling pre-requests from the Client-Proxy. When the Server-Proxy receives a pre-request, it will check the availability of the required lock on the resource specified using the information contained in the pre-request. According to the results from checking, the Server-Proxy will send the Client-Proxy a response which contains either a token if successful or negotiation information if unsuccessful.

Locking Protocol Basics

The client's WS requests can be simply categorized into two groups, read operations and write operations. Thus a reservation based resource locking protocol is proposed for resource coordination on the provider side. In order to support the locking, the following elements are defined.

Locks. A lock on a resource gives the owner of the lock access to the resource. Four types of locks are defined in the proposed locking protocol.

S: Read lock

X: Write lock

IS: Intent Read lock

IX: Intent Write lock

Besides the X (exclusive locks) and S (shared locks) locks which are commonly used for write and read operations respectively, the IS and IX locks are also introduced in the locking protocol. For simplicity, the X and S locks are called execution locks; and IS, IX locks are called intent locks. Intent locks are used for tentative reservation and clients should hold intent locks before they can obtain execution locks. Unlike the intent locks in databases systems where they work at a coarser data granularity than the related S or X locks, the intent locks in our proposed locking protocol are used for the same resource or object which the related execution locks are for. And the purpose of the intent locks is just an indication of the reservation. To be more

specific, IS locks are given to those who intent to read; and IX locks are given to those who intent to write. The reason for introducing intent locks in the approach is that there is still uncertainty in a client's behavior. Thus rather than giving stronger X or S locks to clients directly; introducing light weight intent locks for intent reservation will reduce the cost of the cancellation of X or S locks.

Lock compatibility. Since four types of locks are defined, the lock compatibility rules should also be defined to guide the use of locks. Here the lock compatibility describes what locks can be issued when other locks already exist on the requested resources. Therefore, the compatibility table, as shown in Table 4-1, is not symmetric. According to the first row, an IS lock on a resource can be issued when there are already other IS locks, S locks, or IX locks existing; however, it cannot be issued when there is an X lock on the resource. IX locks are compatible with IS and IX locks, but not S or X locks; the same rule applies to X locks. S locks are compatible with IS locks, IX locks, and other S locks on the same resource except for X locks.

Table 4-1. Lock compatibility among different locks

New Lock	IS	IX	S	X
IS	√	√	√	×
IX	√	√	×	×
S	√	√	√	×
X	√	√	×	×

√ = compatible, × = incompatible

Resource time-lock table. For each resource, a time-lock table is assigned. A resource time-lock table presents information about the locking status of the resource at different time periods. Thus there are three basic elements in a resource time-lock table. They are: the resource,

lock-objects, and time periods. For each time period (a time unit), there is a lock-object associated with it. A lock-object controls how locks on this resource can be assigned within this time unit. In other words, every lock-object functions as a lock manager that manages all locks for this resource but only at the associated time unit. Since a time unit is indivisible, it is reasonable to assume that a client can only book either S or X lock, IS or IX lock for the same time unit. As for the actual value of a time unit, it depends on what kind of services the service provider provides, since different services require different precision of time for operation. For example, in a simple weather report service, the data required by the service client may not be large, thus the time unit value could be defined in the level of seconds; however, in a service which requires large amount of data to be operated, the value for the time unit could be several minutes.

Locks are obtained through pre-requesting. The Client-Proxy composes pre-requests which contain information such as resource name, lock type, time period, etc. If a client plans to do a read operation, it needs two locks: an IS lock and an S lock. The Client-Proxy first requests for an IS lock indicating that the service client intends to request for a read service. Later on, when confirmed of the read operation from the client, the Client-Proxy then sends another pre-request to obtain an actual execution lock, the S lock. And the S lock can only be given when an IS lock is already been held. So does it apply to write operations which require IX and X locks. The proof of having a lock hold is a token. When the Client-Proxy sends a pre-request booking for an IS lock, if applicable, a unique token will be assigned. Then, the Client-Proxy can use the token to book an S lock and get another token for the actual WS request.

Lock Overriding and Compensations

Since locks are associated with time units, there are conflicts in lock reservations. For example, Client A comes to book an S lock for the time period of T2. Unfortunately, an X lock at

the same time period T2 has already been assigned to Client B. Could Client A get the lock? And under what condition might it get the lock? These are the questions to be investigated in this section.

Before moving on to the answers, it is necessary to review the cooperative behavior. Three styles of behavior are proposed. In the last two cooperative behaviors, clients can contribute their effort directly to the resolution of the conflicts, since they are e aware of their friends' behaviors when conflicts happen by the use of friend lists. However, in the first one, the Closed Consumer-Provider Cooperation, clients are unaware of each other. Conflicts are solved by only the provider. Thus, compensations should be provided to those clients who unexpectedly but cannot avoid to lose their service/resource. So, this section will discuss the conflicts resolution for clients in the first cooperation scenario.

Table 4-2. Lock overriding policy

Priority	P_n	P_{n-1}	P_{n-2}	P_{n-3}
P_n	×	Level1	Level2	Level2
P_{n-1}	×	×	Level1	Level2
P_{n-2}	×	×	×	Level1
P_{n-3}	×	×	×	×

Level1 = Level 1 overriding available
Level2 = Level 2 overriding available
× = overriding unavailable

If a conflict for a lock happens, there will be two kinds of results. Result one: the lock still belongs to the one who first owned it; the one who came later cannot get the lock. Result two: the lock goes to the one who came later; the first owner gets compensations for losing the lock. As clients can be grouped by their priorities, a priority-based two level lock overriding policy is proposed to deal with conflicts. Level 1 overriding means intent locks can be

overridden; Level 2 overriding means both intent locks and execution locks can be overridden. Suppose there are n different priorities, listed as P_1, P_2…, P_n ($n \geq 2$). P1 is the lowest, and P_n is the highest. Thus, P_k clients can override intent locks that belong to P_{k-1} or lower priority clients. And P_k clients can override both types of locks that belong to P_{k-2} or lower priority clients. Please see Table 4-2 for a detailed listing.

According to the proposed priority-based two level lock overriding policy, lower priority clients are the most unfortunate since they always lose locks when they have lock conflicts with higher priority clients. This is reasonable since lower priority clients pay less than higher priority clients for the services. But in order to maintain good service consuming experiences, compensation policies are needed to prevent low priority clients from continually losing locks.

Accordingly, a two level compensation policy is proposed. Clients who lose an intent lock will be guaranteed no loss of a lock in case of the next conflict. However, things will return to normal when the given guarantee is consumed. Similarly, clients who lose an execution lock will also be guaranteed the possession of locks. The difference from level 1 compensation is that this guarantee can be used for the next two conflicts.

Re-negotiation

Negotiations take place during the WS contract stage. WS consumers and WS providers negotiate a service contract which includes various aspects of service quality, price, etc. In the proposed cooperative approach, a re-negotiation also happens during service consumption. This will create more communication opportunities for both clients and providers, leading to enhanced experiences of service consumption.

As there are different cooperation scenarios, the negotiation protocol may vary. In this section, the negotiation protocols for the first two cooperation scenarios are discussed. The third scenario will be the future work.

Re-negotiation in the Closed Consumer-Provider Cooperation

A lock overriding policy is introduced to deal with lock conflicts. However, it can only solve the problems when a high priority client comes after a low priority client. Let's still take a look at the example used in the last section. Client A comes to book an S lock for the time period T2. Unfortunately, an X lock at the same time period T2 has already been assigned to Client B. This time, Client A's priority is lower than Client B's, and then the lock overriding policy cannot help. So what will happen to Client A? Will the client get an error message indicating that services are unavailable? Or will it get no response at all? The cooperative approach dedicates to providing WS clients more options for services and high availability of WS rather than an error message without any help. The re-negotiation focuses on the time change. As Client A cannot override Client B's lock, the Server-proxy will suggest Client A the next available time period for its request. Then the Client-Proxy will notify Client A about the event of time change. The client will decide if it agrees with the suggestion or would like to continue the negotiation.

Re-negotiation in the Open Consumer-Provider Cooperation

When service consumers are in an open cooperation scenario, they can contribute more efforts to the negotiation. As a client has provided its friend list to the service provider, it can be notified if a conflict happens between itself and one of its friends. Now the re-negotiation begins. After being informed of each other's request, both the client and its friend will evaluate how important their own requests are. Rankings of their requests then are sent to the provider to make the decision of who wins the lock by comparing the two rankings. In order to avoid cheating or dishonest rankings, penalties should also be proposed in the re-negotiation. However, in this research, the penalty is not implemented. A more decentralized way for re-negotiation is to let service clients negotiate themselves. This requires the ability of clients to exchange messages at runtime.

If communication between clients is enabled, more interesting approaches are possible. One example is to let the clients bid for resources in case of conflicts. This will involve the clients to actively participate in the re-negotiation and it can solve problems such as lower priority clients denying the conflict solutions provided by the provider.

The cooperative approach requires service participants to exchange and share information at runtime. This information includes the clients' workflows, the state of the resource utilization of the service provider, and etc. All these are sensitive data which are not supposed to be released to third parties. Furthermore, a trust mechanism is necessary to ensure the cooperation in the Open & Decentralized scenario, in which service clients are aware of each others' actions when resources conflicts happen and are required to rank their own requests as well as those of their components'. As a result, trust management and ensuring data security and privacy protection are important issues in the cooperative approach. However, this research is just a beginning on exploring cooperation in WS and thus currently only focusing on basic ideas. Whether this proposed approach can improve the experience of WS consumers in terms of performance is the main goal at present. Therefore, security and trust are not discussed in this scope.

CHAPTER 5
EXPERIMENTS

This chapter evaluates the proposed cooperative approach with experiments on stationary machines. The main goal of the experiments is to study the impact of using cooperation for service consumption with respect to performance and service availability. The evaluation is divided into two phases:

Phase 1: evaluating the Closed Consumer-Provider cooperative behavior. In this phase the cooperative approach is evaluated using basic services.

Phase 2: evaluating the Open & Decentralized Consumer-Provider cooperative behavior. In this phase, the cooperative approach is evaluated within E-Commerce scenarios.

The client-perceived response time is used as the parameter measuring performance. It is measured in milliseconds from the time the service client actually sends a request to the point when it successfully receives the response from the service provider. The response time includes the server execution time, the transmission time, and the waiting time for processing. And service availability is measured by the ratio of successful operations.

The first section presents the experiments in phase 1using an abstract service. Section two presents experimental results of phase 2 using an E-Commerce style of service.

Phase 1 Experiments

In Phase 1 experiments, the cooperative approach is evaluated within the first cooperation scenario.

Experimental Setup

- A stationary machine as the server: It is a HP xw6400 Workstation, which has the following hardware and software configuration: Intel(R) Xeon(R) CPU, 5140@ 2.33GHz, 1.98 GHz, 2GB RAM; the operating system is Windows XP, and the server is the Netbeans built-in application server Glassfish V2.

- Lab machines as clients: Two Intel(R) Core(TM)2 6600@2.40GHz CPU; 2GB RAM. Software configuration: the operating system is Linux 2.6.24.7; Java 6 is installed. The client machines and the server machine are connected to the same network.

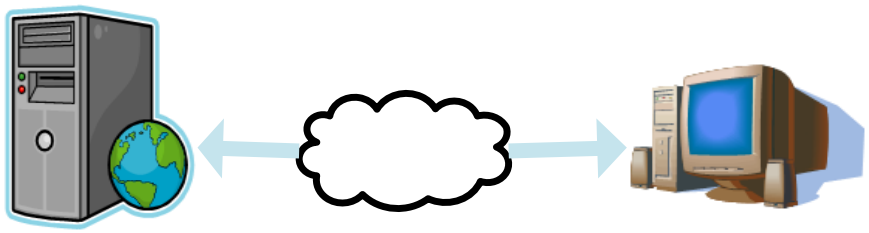


Figure 5-1. Setup of the experiment system

A test bed was built in Java 6. For simplicity, the SOAP based service is atomic. It provides three functions; two read operation (readA, readB) and a write operation (writeA). The resources on the services are two integer variables: variable A, and variable B.

In the test bed, the Server-Proxy is implemented as a component residing on the server side. The Client-Proxy object resides in the service client's application so that every time the client's application starts, a client-proxy will be instantiated. The Client-Proxy uses SOAP messages to communicate with the Server-Proxy. For proxy-transparency, the Client-Proxy, the Server-Proxy, and the Read-Write WS all have the same functional interface. In the implementation of the locking protocol, the time unit is set at 1 second. The replacement algorithm used for the cache in the Client-Proxy is LRU.

Single-Client Workloads

This experiment aims to investigate the overheads caused by the cooperative approach and the gains in term of performance.

In this experiment, a set of 100 identical reads and a set of 100 identical writes were conducted using both the cooperative approach and the conventional approach, in which the

service clients have direct communication with the service provider. The think time, which is the time interval between two requests in both approaches, is one second. And the time out value is 1 second, which means that if the server resource is not available immediately at the time the request arrives, the client will wait for the server resource to be available for 1 second. The measurement in the experiments is the average client-perceived response time.

In order to determine the overheads, how the overheads are distributed, and how much can be gained from using prediction and cache respectively, the prediction accuracy and the caching percentage are adjusted for the experiments. The caching percentage means the percentage of the results that are available in the cache. The prediction accuracy rate refers to the ratio of the requests (in a workflow) which are correctly predicted by the prediction model.

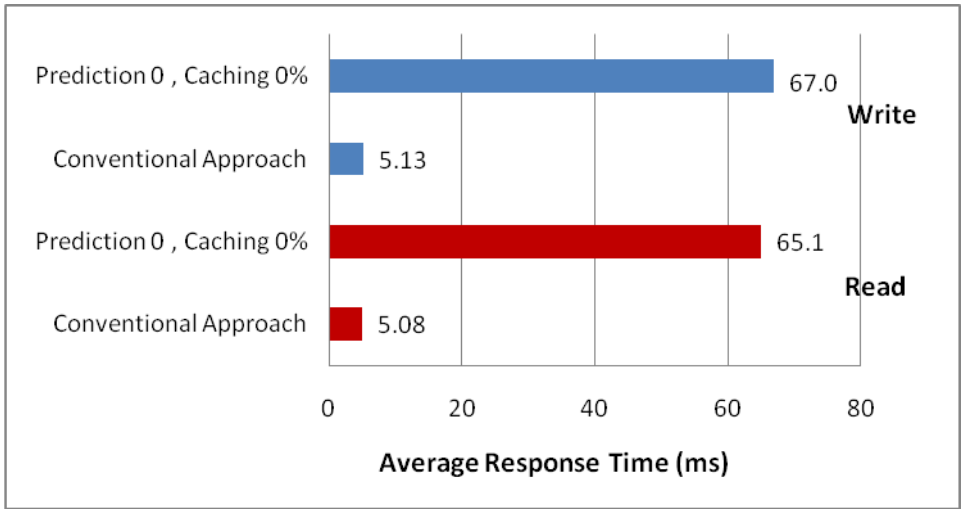


Figure 5-2. Overheads of the cooperative approach in the worst situation

Figure 5-2 presents the overheads caused by the cooperative approach in term of average client-perceived response times (in milliseconds) of repeated read operations and write operations. In the cooperative approach, the prediction accuracy is set at 0. And the percentage of

the cached data in the workloads is set at 0% in order to get the overheads caused by the cooperative approach in the worst situation. As Figure 5-2 shows, the overheads of both read and write operations are relatively large compared to the conventional approach, 65ms and 67ms respectively.

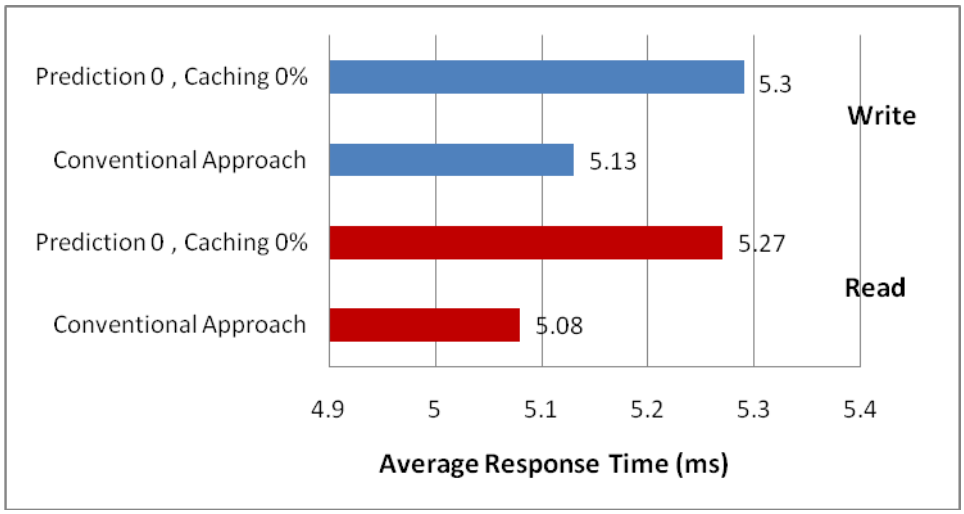


Figure 5-3. Gains from only prediction when caching is worst

Figure 5-3 shows the average client-perceived response times of 100 read operations and 100 write operations in both approaches. In the cooperative approach, the prediction accuracy is set at 1 and the caching percentage remains 0% in order to find out the gains from only the prediction component, as well as the overheads mainly caused by cache. It is reasonable to measure the cache overheads in this way, because when the prediction accuracy rate is 1, all the pre-requesting is completed prior to actual requests; thus the pre-requesting time is not included in the client-perceived response times. As indicated by Figure 5-3, the average response times of both read and write operations in two approaches are similarly small. This reflects that when fully functioned, the prediction component cuts most of the overheads from the worst situation.

From another angle, it can be concluded that the cache causes only a minor overhead, around 0.2ms for both operations.

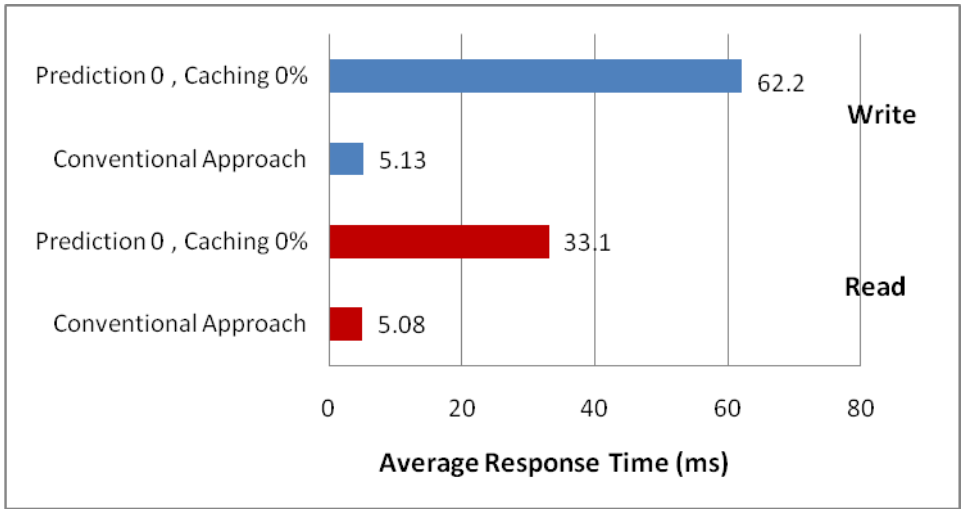


Figure 5-4. Gains from only caching when prediction is worst

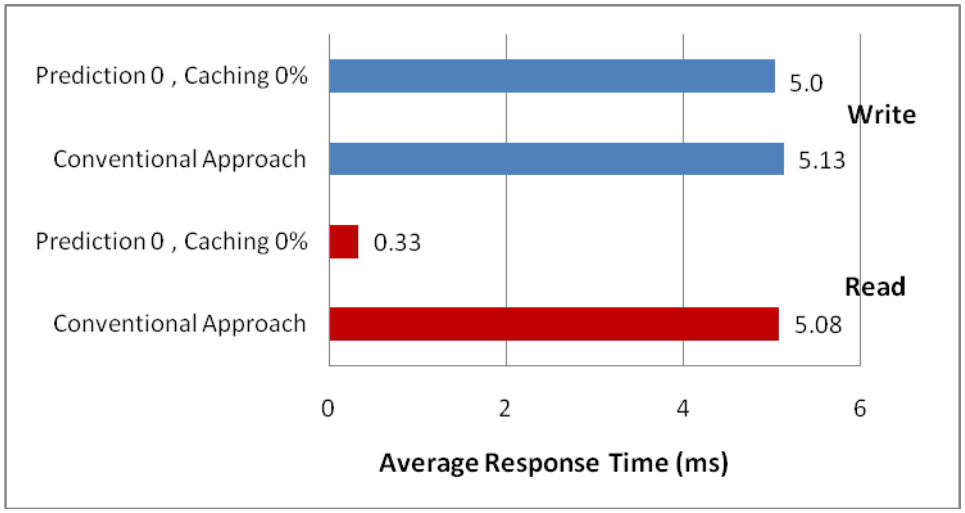


Figure 5-5. Gains from the cooperative approach in the best situation

When looking at Figure 5-4, the result is quite different from the previous experiment. In this experiment, the caching percentage is 100% and the prediction accuracy is 0. This zero

prediction accuracy implies that the Client-Proxy has to do pre-requesting at the time the client actually sends a request. As a result, the pre-requesting time is included in the client-perceived response times. From the figure, response time of read operations in the cooperative approach is half of the one in the worst situation, whereas the response time of write operations has no improvement. This result shows that the pre-requesting is time costly. When prediction is the worst, cache cannot help much with respect to response times.

Figure 5-5 shows the gains from the cooperative approach in the best situation, where both the prediction accuracy and the caching percentage are 1 and 100% respectively. As indicated by the figure, the result is much better. When both prediction and cache function perfectly, the average response time of read operations is 0.66ms, close to 0. This is because both the costly pre-requesting and the sending of SOAP messages are avoided. As for the write operations, the average response time finally comes close to the one in the conventional approach.

Multi-Client Workloads

This experiment aims to investigate the performance of the cooperative approach when resource conflicts exist.

Ten clients with different priorities for the cooperative approach are used. Of the 10 clients, 3 clients have the highest priority, 2 have the lowest, and the others have the medium priority. 10 clients are built for the conventional approach; and these 10 clients are of equal type with no priorities specified. All the clients have the same workflow which consists of 10 operations, 7 reads and 3 writes. The think time between each operation in the workflow is 2.5 seconds. The timeout value for the WS call is 1 second. The arrival rate of the clients is 1/2 per second, which means the time interval between each client to start is 2 seconds. In both approaches, every request (including both read operations and write operations) will require the resource to be held

for 1 second. The prediction accuracy and the caching percentage in the cooperative approach are set at 1 and 100% respectively.

Figure 5-6 shows the average response time per request of each client in both approaches. As can be seen in the figure, the response time is greatly reduced by using the cooperative approach.

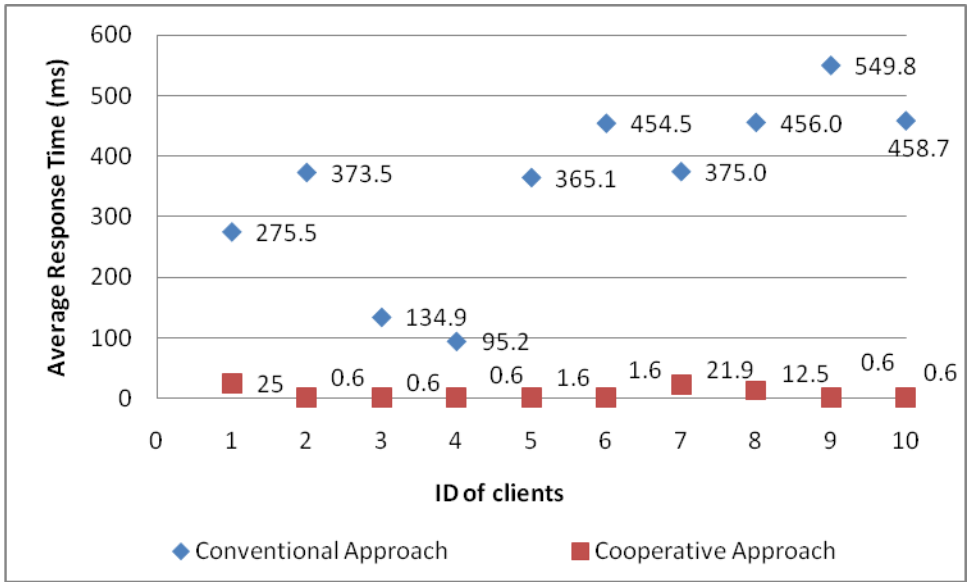


Figure 5-6. Average response time of service clients

Table 5-1 shows some other parameters in both approaches.

Table 5-1. Some parameters in both approaches

Parameters	Conventional approach	Cooperative approach
ServiceAvailablity (percentage)	79%	100%
NegotiationRate (percentage)	N/A	28%
AvgResponseTime of reads (ms)	382.1	4.7
AvgResponseTime of writes (ms)	272.4	9.9

ServiceAvailability refers to the percentage of the operations that consume the services successfully. In the conventional approach, the Availability is only 79%, which means 21% of

requests encounter service unavailability. In contrast, the service Availability in the cooperative approach is 100%.

NegotiationRate refers to the percentage of the operations that encounters a re-negotiation with the service provider about the change of the operation time. This is only applicable to the cooperative approach since there is no re-negotiation process in the conventional approach. The NegotiationRate is relatively high, with a value of 28%. This could explain why the ServiceAvailability is 100% in this situation; it is because when there is a conflict of the server resource, the Server-Proxy can initiate a negotiation with the Client-Proxy about the change of the operation time.

The other parameters are regarding the average response time. As shown in the table, the cooperative approach provides great improvement in the response time of either type of operations, and at the same time it maintains high service availability.

Phase 2 Experiments

In this phase of experiments, the goal is to investigate the performance of the proposed cooperative approach using the second cooperation scenario: Open & Decentralized Cooperation Scenario.

Experimental Setup

The hardware and software configurations are almost the same as Phase 1 experiments except for the end service. Instead of evaluating the proposed cooperative approach using the abstract service as used in previous section, an E-Commerce service was built to simulate a more realistic scenario: the on-line shopping scenario. This E-Commerce service was built using the REST style as it does not require much effort in developing an application. Moreover, I also like to see if the results are consistent with previous ones using different style of WS.

Table 5-2. Service Operations

Operation Name	HTTP Command	Semantics
ProductDetails	GET	Get the details of one product
ProductsDetails	GET	Get a list of products
Bestsellers	GET	Get a list of products that sell best
CreateOrder	POST	Create a new order for one product
ViewOrder	GET	Get the details of an order

The WS have several service operations as shown in Table 5-2. Resources on the provider side are products and orders. In the experiments, only ProductDetails and CreateOrder operations are used to simulate read and write operations. The RESTful WS use HTTP as the communication protocol, as well as the application protocol. This is because the service semantics are along with the HTTP commands. For example, a GET command for the productDetails operation means to retrieve the detailed information of a product from the service provider. A POST command for the CreateOrder operation refers to creating a new order for a product. Data are stored in the Netbeans built-in Apache Derby database. The product table contains 629 records. Both the WS and the database server stay on the same machine.

The Client-Proxy resides on the client's side and provides the same interfaces as the RESTful WS. In this set of experiments, the Client-Proxy is simple. One assumption is that the Client-Proxy has the client's workflow information. The replacement algorithm used for the cache in the Client-Proxy is LRU.

The Server-Proxy stays on the provider side. In this experiment the Server-Proxy functions as the resource coordinator which processes pre-requests from client-proxies. The Server-Proxy also keeps all the clients' friends lists for re-negotiation.

As TPC-W (Transaction Processing Performance Council – Web) E-Commerce benchmark [39] defines, there are three types of web workloads: browsing, shopping, and ordering. The three workloads are composed of different proportions of read and write operations. Table 5-3

shows the detailed the distribution of read and write operations in the three workloads proposed in TPC-W.

Table 5-3. Proportion of reads and writes in TPC-W workloads [39]

Workload	Percentage of Reads	Percentage of Writes
Browsing	95%	5%
Shopping	80%	20%
Ordering	50%	50%

Experiments are conducted with single client and multiple clients. Below are several assumptions and settings:

- All the clients simulated in the following experiments have exactly the same workflow for each type of the workloads (browsing, shopping, and ordering).
- For each type of the workloads, there are 50 requests.
- The time interval between two requests in all the workflows is 1 second.
- The TTL parameter is set 120 seconds for the cache in the Client-Proxy. As a result, no invalidation happens since all the requests will be finished within 60 seconds.
- For multiple-client experiments, there are 5 clients simulated. Each client starts right after each other and the time interval between two clients to start is less than 3 seconds.
- All the resources in the server are small and have the same size, around 600 bytes.

Overheads and Gains

This set of experiments aims to find out the basic overheads and gains of the cooperative approach. A single client was simulated with different settings and different workloads. The average response time was taken as the measure of the performance.

Figure 5-7 shows the overheads of the cooperative approach compared to the conventional approach in terms of the average response time per request. As shown in the figure, the overheads of the write operations in the cooperative approach are quite large in all of the

settings. This is as expected since every write operation has to route through the Client-Proxy and then the Server-Proxy to reach the server; neither the caching nor the prediction can help much when no resource conflicts exist. For the read operations, the performance varies according to the settings. In the worst situation where both caching and the prediction have the worst performance, the overhead is large. However, with the help of perfect caching, all the overhead is cut off and the response time is even smaller than that in the conventional approach. The prediction performs slightly worse than the cache. This also means that the prediction causes more overheads than the cache. And this also applies to the write operations. In the best situation, the cooperative approach has a huge gain of performance for the read operations. These results are similar to the one in the phase one experiments.

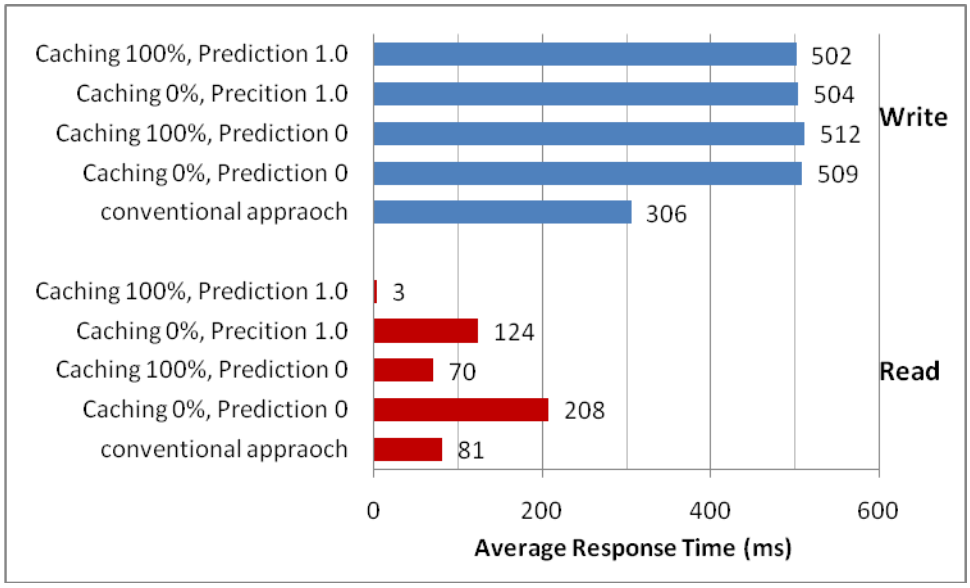


Figure 5-7. Average response time with different settings in the browsing scenario

Figure 5-8 and Figure 5-9 show the overheads and gains in the shopping and ordering scenarios. The results are similar to that in the browsing scenario.

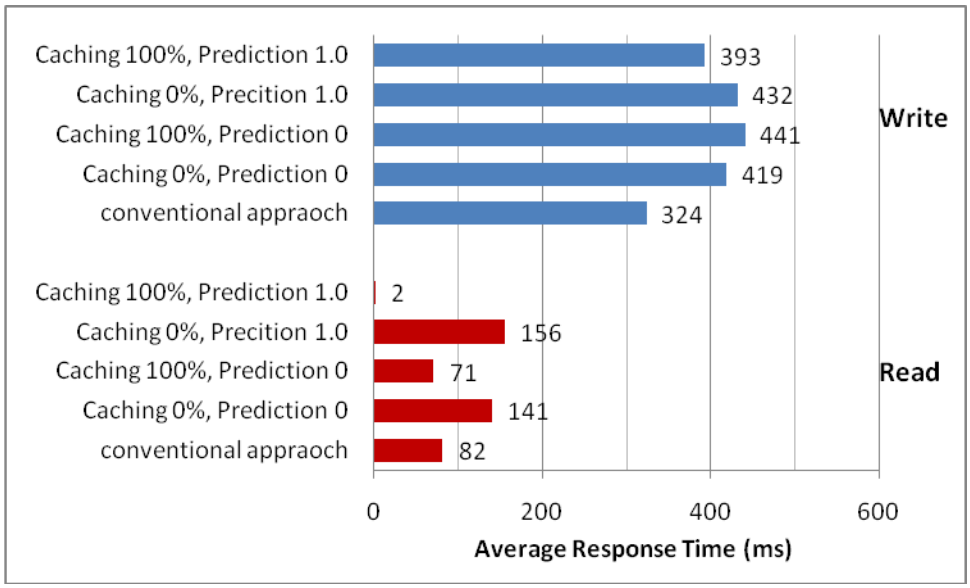


Figure 5-8. Average response time with different settings in shopping scenario

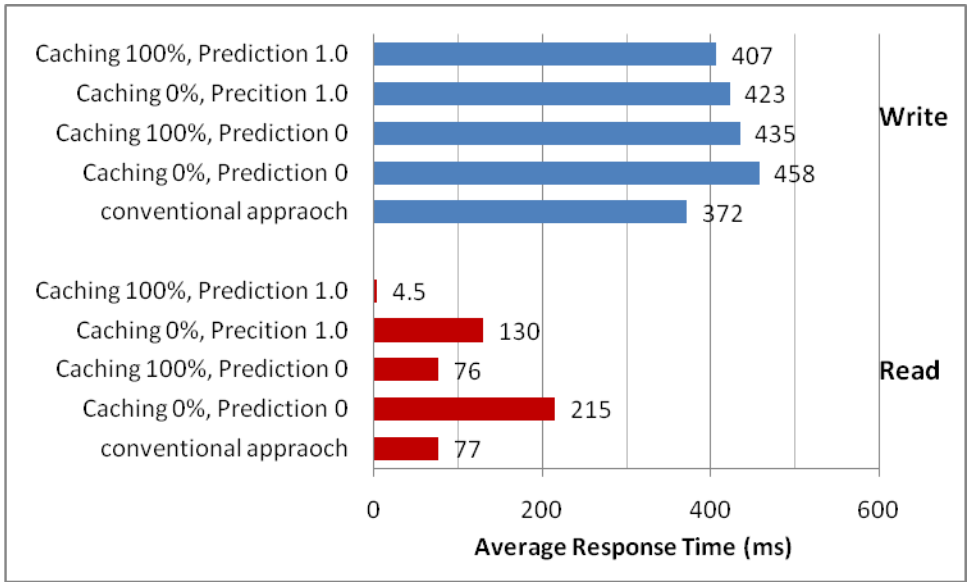


Figure 5-9. Average response time with different settings in ordering scenario

Impact of Caching on Performance

These experiments aim to investigate the impact of caching on performance. To control the caching percentage, a portion of the results were pre-cached in the cache. The prediction accuracy rate is set at 1 so that it can cause as little overhead as possible. The experiments were conducted using a single-client scenario and a multi-client scenario. In the multi-client scenario, 5 clients were simulated. In order to avoid overloading for the server, the ordering scenario was not simulated for the multi-client experiments.

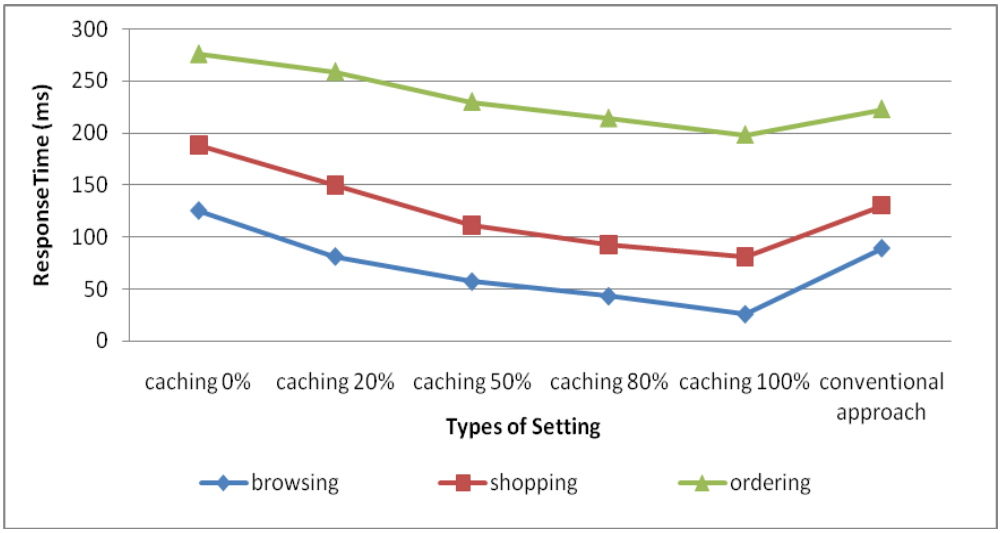


Figure 5-10. Impact of caching on performance in the single-client scenario

Figure 5-10 shows the results of the single-client experiment. The data of the 0% and 100% caching from the previous experiments are also added to the graph. As is shown, the average response time increases as the percentage of caching increases. Among the three workloads, browsing has the best performance in all the settings. The ordering workload comes at the last place. Compared to the conventional approach, the cooperative approach performs better only when the caching percentage is above 50%.

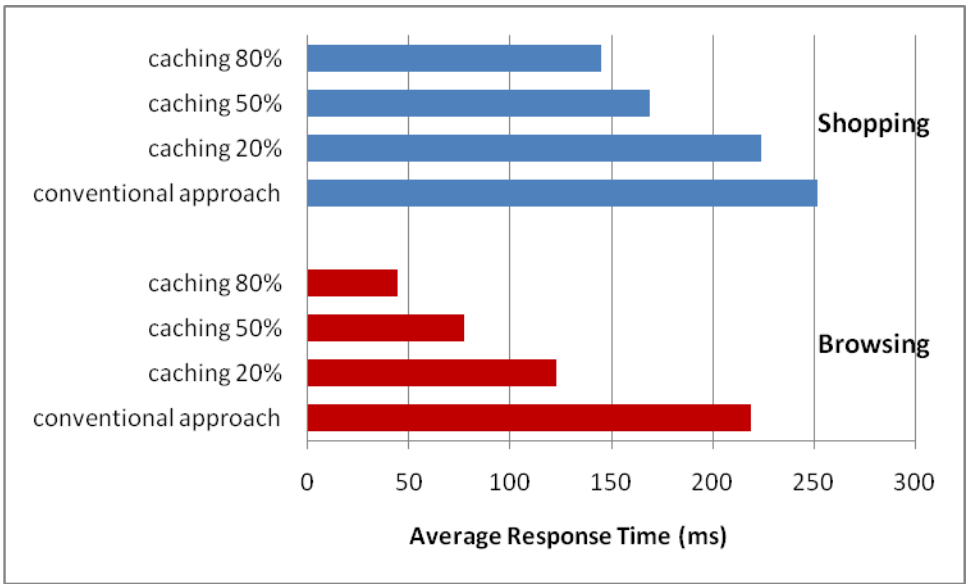


Figure 5-11. Impact of caching on performance in the multi-client scenario

Figure 5-11 shows the result of the multi-client scenario. The result is a little different from that in the single-client experiment. The conventional approach performs worse when the percentage of cached results is above 20%.

Impact of Prediction on Performance

In this section, the impact of prediction on performance is evaluated. The caching is 100%. The experiments were conducted using a single-client scenario and a multi-client scenario.

Figure 5-12 shows the average response time of the single-client experiment with different prediction accuracy rates. The result is similar to the one in the experiment on caching. Again, as the prediction accuracy rate increases the performance increases. Compared to the conventional approach, the cooperative approach is better when the prediction accuracy rate is above 0.5 in all the three workloads

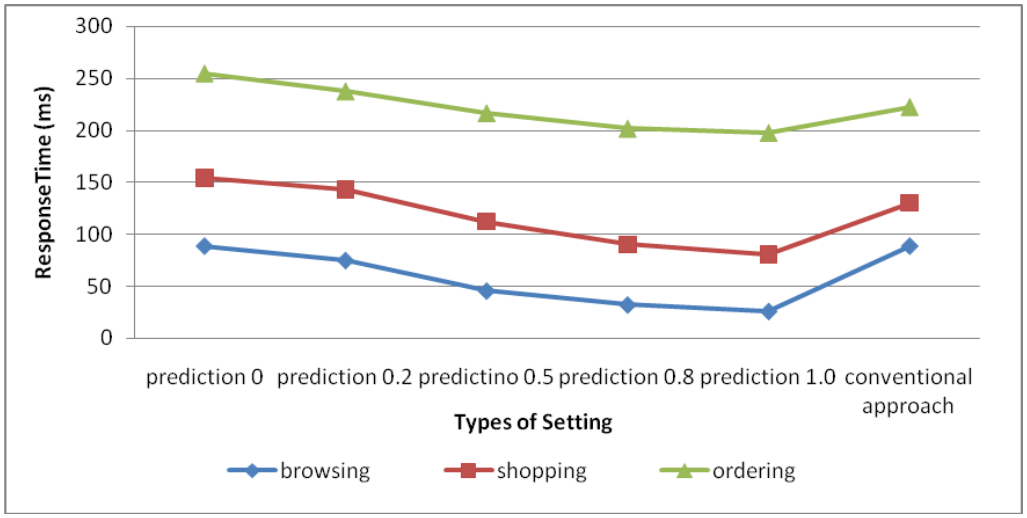
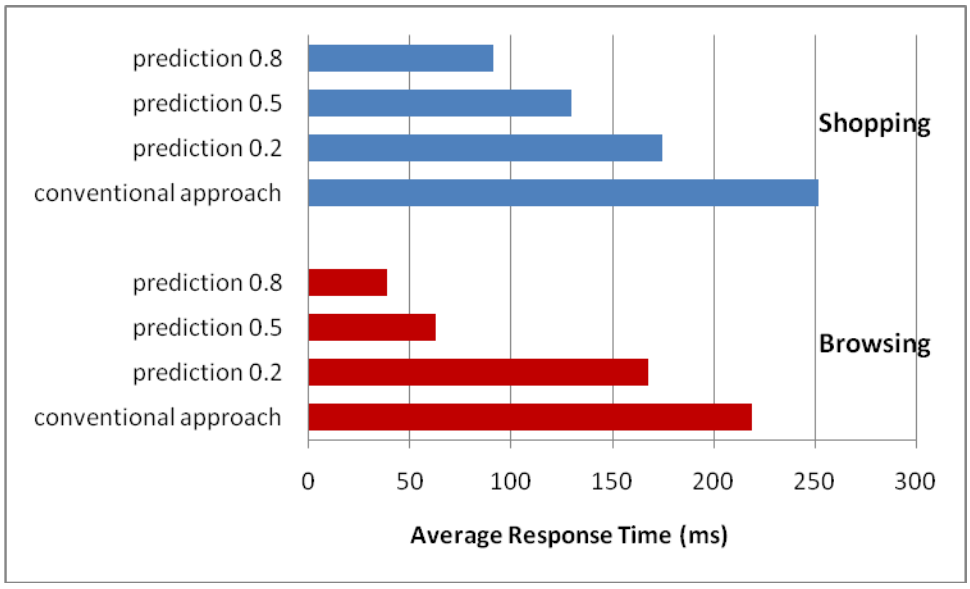


Figure 5-12. Impact of prediction on performance in the single-client scenario

Figure 5-13 shows the result of the multi-client experiment. Again, the cooperative approach outperforms the conventional one when the prediction accuracy rate is above 0.2.



.

Figure 5-13. Impact of prediction on performance in the multi-client scenario

Conclusion

Experiments in this chapter evaluated the cooperative approach using two cooperation scenarios: the Closed & Centralized Consumer-Provider cooperation and the Open & Centralized Consumer-Provider cooperation. In both scenarios, the caching and the prediction perform consistently. Caching works better for read operations and prediction is crucial to write operations. The improvement in performance in the cooperative approach is more obvious when there are many clients competing for the resources on the server.

Performance is usually a factor that service clients care most about during service consumption. The simple prediction model and caching in the cooperative approach can help improve WS performance by reducing response time. By using the simple prediction model, the Client-Proxy can inform the service provider of the client's next request so that the server can prepare ahead of time for the client's requests.

Availability is another important QoS attribute for service clients since they expect the services to be always there for serving requests. To assure service availability, the cooperative approach employs a reservation-based locking protocol for resource coordination and a re-negotiation strategy. By resource locking, every service client will get those server resources promised when it goes to the server at the agreed time for processing. And it will not be interrupted by other clients' requests since the resources are already reserved for its own use. Re-negotiation further enhances the availability of services. For example, if a specific time required by the client is not available on the server due to locked resources, the service provider will initiate a negotiation process with the client by suggesting another available time to the client. And in the second cooperation scenario, negotiation between friends is possible. They can negotiate themselves to solve the conflicts on resources by ranking requests and the service provider even does not need to provide solutions all by itself. This allows clients to have more

choices consuming services rather than just sending an error message to clients indicating that requests cannot be accepted.

Reliability refers to how well the service provider can provide correct WS. In the cooperative approach, every request that has been pre-processed will get a unique token prior to service consumption. So the request and its response are correlated by this unique token. This ensures a guaranteed message delivery.

CHAPTER 6
EVALUATION WITH MOBILE DEVICES

Mobile and nomadic devices such as cell phones and PDAs have evolved in recent years from resource constrained appliances to highly connected and increasingly powerful devices. This in turn has led to the development of standardized platforms such as java micro edition, Android and the IPhone OS which enable third party developers to build applications fairly easy. However, standalone applications for the mobile/nomadic device have been shown to be of limited use to users. Instead, users are more interested in using such devices to seamlessly access the IT resources through the internet. Thanks to the widespread acceptance of Web Services, it is fairly easy to access resources/services from a mobile/nomadic device. However, by relying on external resources and services, the QoS is of the upmost importance. Especially if a mobile/nomadic device is engaged in the execution of a workflow, it is vital to keep service latencies at a minimum. Thus how the cooperative approach performs on mobile devices is a very interesting question in this research. This chapter presents the basic evaluation of the proposed cooperative approach used on mobile devices.

Experimental Setup

In these experiments, four different machines are used. The detail of the hardware and software configuration is as the following.

- The server: These experiments use the same RESTful service in the previous experiments. It is hosted on an iMac which has Intel Core 2 Duo 2.66 GHz, 4 GB RAM. This machine is connected to the network via 100 Mbps Ethernet. The HTTP server is Glassfish V2.
- The Server-Proxy is on a HP xw6400 Workstation which has Intel(R) Xeon(R) 5140@2.33 GHz, 2 GB RAM. The OS is Windows XP. This machine is connected to the network via 100 Mbps Ethernet.
- The Client-Proxy resides on a MacBook which has 2.4 GHz Intel Core Duo, 2 GB RAM. This machine is connected to the network via 100 Mbps Ethernet.

- The clients are two smart mobile devices. One is an IPod Touch. The other is a G1 Google smart phone with Android software stack. Both of these two devices are connected to the network via WiFi wireless network.

The experimental setup is shown in Figure 6-1. Rajitha Bakthula developed the client application for the IPod Touch experiments.

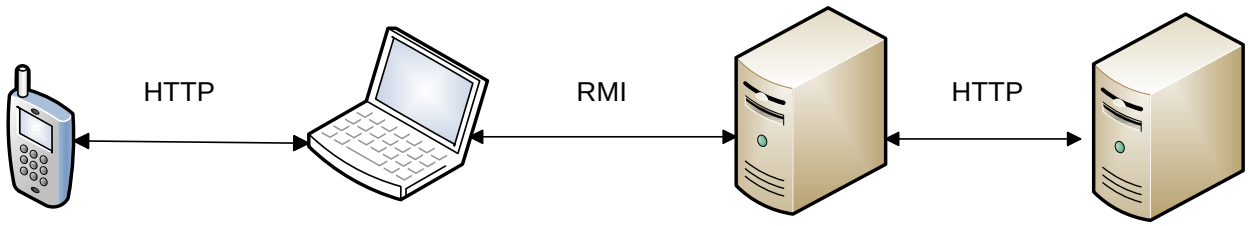


Figure 6-1. The experimental setup

Experiments on the Caching and Prediction

In this part, the previous experiments were repeated on two smart mobile devices. One is the G1 Phone and the other is the IPod Touch. Again, the TPC workloads are used. For each type of workflow, there are still 50 requests in total. The time interval between each two requests is 1second. The TTL parameter for the cache is still set at 120 seconds and the replacement algorithm used is LRU.

Overheads and Gains

These experiments aim to investigate the overheads and gains of the cooperative approach on the two smart devices. In order to see how the smart mobile devices handle read and write operations respectively, performance of read and write operations is measured separately.

Figure 6-2 presents the overheads and gains for read and write operations in the browsing scenario. For write operations, the overheads of the cooperative approach are relatively large compared to the conventional approach using both devices, no matter how the prediction and the

cache perform. This is as expected because in the conventional approach the client communicates with the REST service directly, while in the cooperative approach the client has to route to the Client-Proxy and the Server-Proxy first, and then reaches the end service. This of course will increase the transmission time. But still, the figure shows that the prediction can reduce almost half of the overhead while caching provides no help at all. For read operations, the overheads are still obvious. In the situation that prediction and caching both have the worst performance, the response time of the cooperative approach doubles in the IPod experiments. And in the G1 phone experiments, the response time increases by one third. This is also due to the routing of the requests. However, the prediction and cache perform differently this time. From the figure, cache reduces more overheads than prediction for read operations.

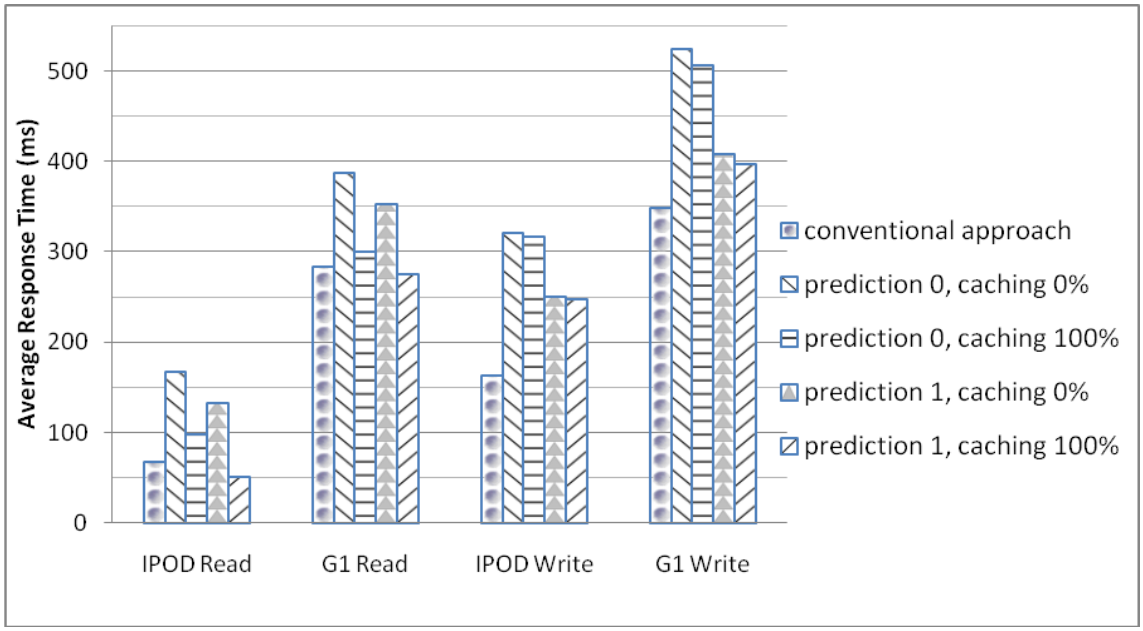


Figure 6-2. Overheads and gains in browsing

Figure 6-3 and Figure 6-4 show the overheads and gains in shopping and ordering scenarios respectively. They have the same pattern as those in the browsing scenario.

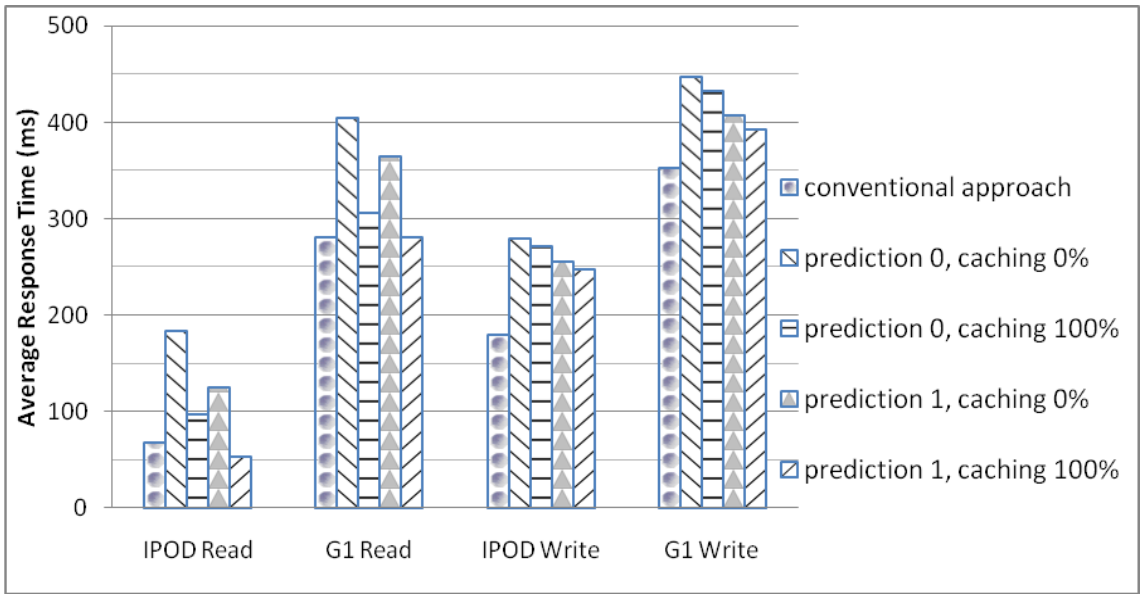


Figure 6-3. Overheads and gains in shopping

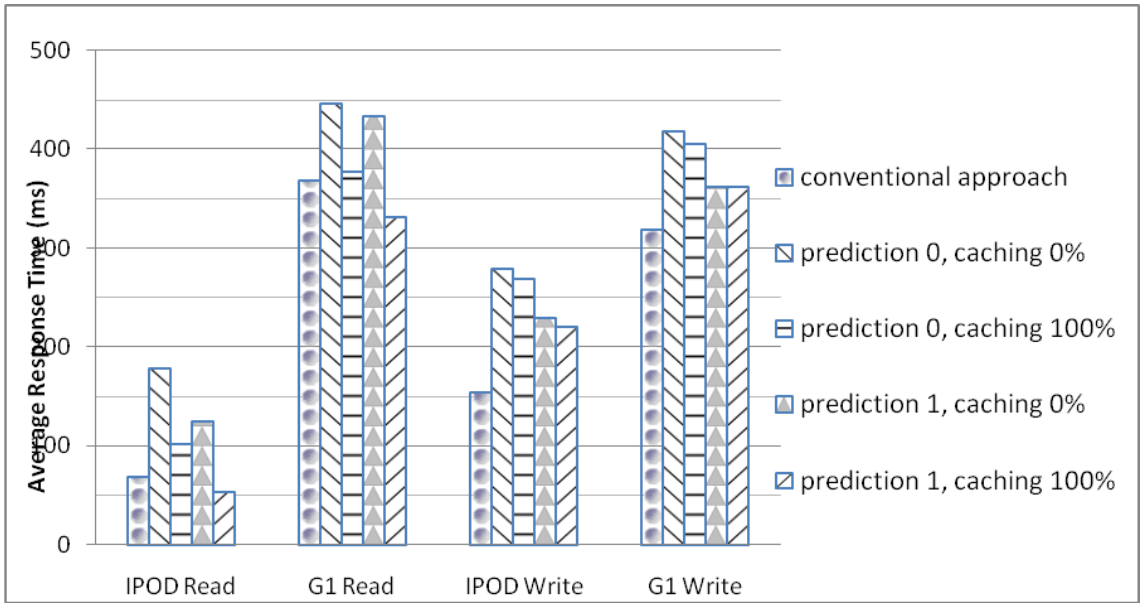


Figure 6-4. Overheads and gains in ordering

Impact of Caching

These experiments aim to investigate the impact of caching on the performance. The caching percentage is adjusted for each run. The prediction accuracy rate in these experiments is set at 1.

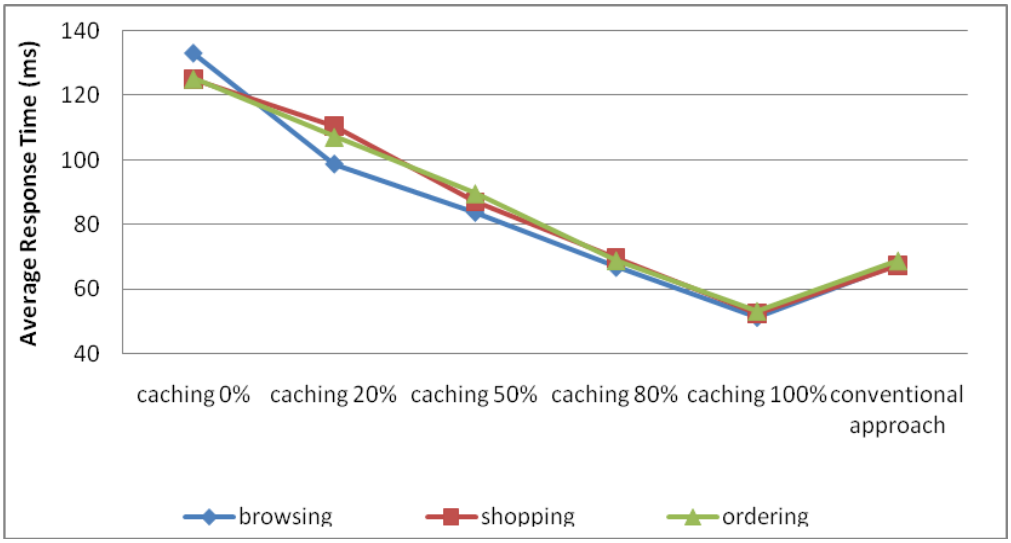


Figure 6-5. Impact of caching on read operations (IPod)

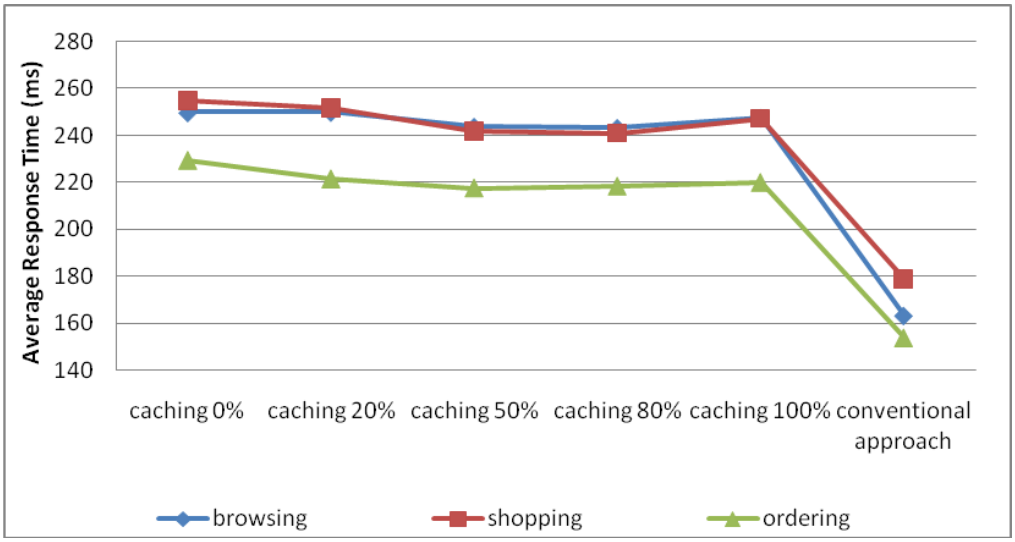


Figure 6-6. Impact of caching on write operations (IPod)

Figure 6-5 shows the average response time of read operations with different caching percentages. The response time in all three workloads shows the same pattern, it decreases as the percentage of cached data increases. And the decrease is almost linear. Among the three workloads, browsing has the best performance in most cases, but the difference is quite small. Comparing to the conventional approach, when the caching percentage is less than 80%, the performance is worse.

Results are quite different for the write operations shown in Figure 6-6. As expected, the cache does not have any impact on the performance of write operations.

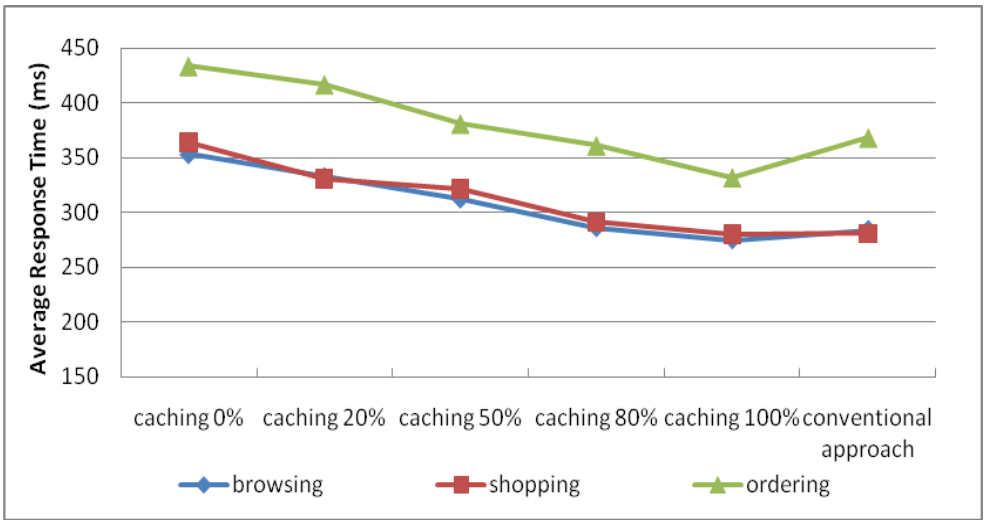


Figure 6-7. Impact of caching on read operations (G1 Phone)

From Figure 6-7, the response times in all three workloads using the G1 phone have similar pattern to those in the IPod experiments. Among the three workloads, ordering has the worst performance. And when comparing the cooperative approach to the conventional approach, the first one outperforms the latter one only when the caching percentage reaches 80%. This is similar to results in the IPod Touch experiments.

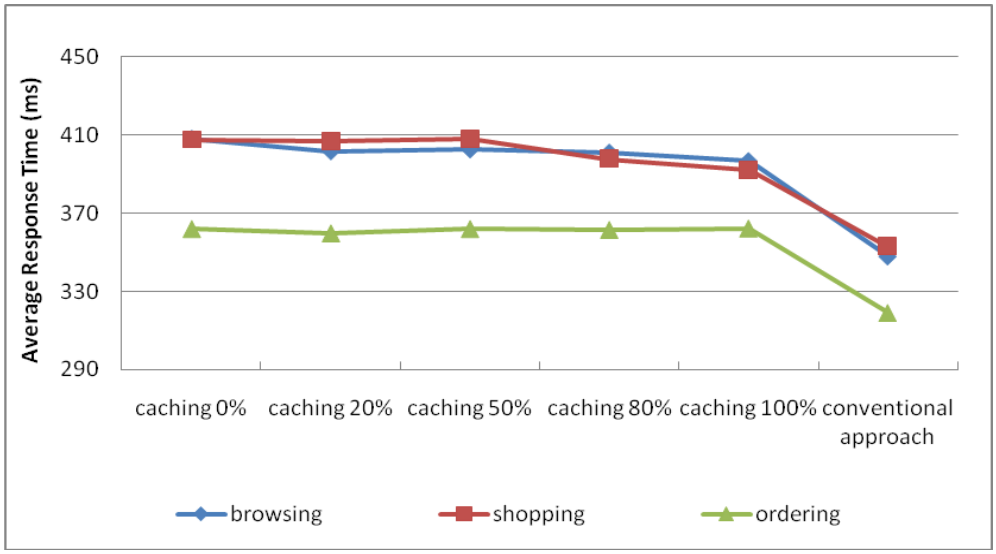


Figure 6-8. Impact of caching on write operations (G1 Phone)

As expected, the cache does not have any impact on the performance of write operations as Figure 6-8 shows. However, an interesting result is that the average response time of write operations is less than that of read operations in the ordering scenario, in which half of the requests are write operations, using both the conventional and the cooperative approaches. This is partially due to the setting in this experiment. The prediction accuracy rate is 1, which contributes a great deal to reducing the latency for write operations. This explains why the write operations perform well. That the read operations perform relatively worse might be due to the implementation of the HTTP mechanism of the G1phone, as well as the low level data serialization and de-serialization processes.

Impact of Prediction

These experiments aim to investigate the impact of prediction accuracy rate on performance. The caching percentage is set at 100%.

Figure 6-9 presents the impact of prediction accuracy rate on read operations using the IPod. As the rate increases, the performance increases as well. This applies in all the browsing,

shopping, and ordering scenarios, which have almost the same performance in different prediction settings. With the help of perfect caching, when the prediction accuracy rate is greater than 0.5, the cooperative approach beats the conventional approach.

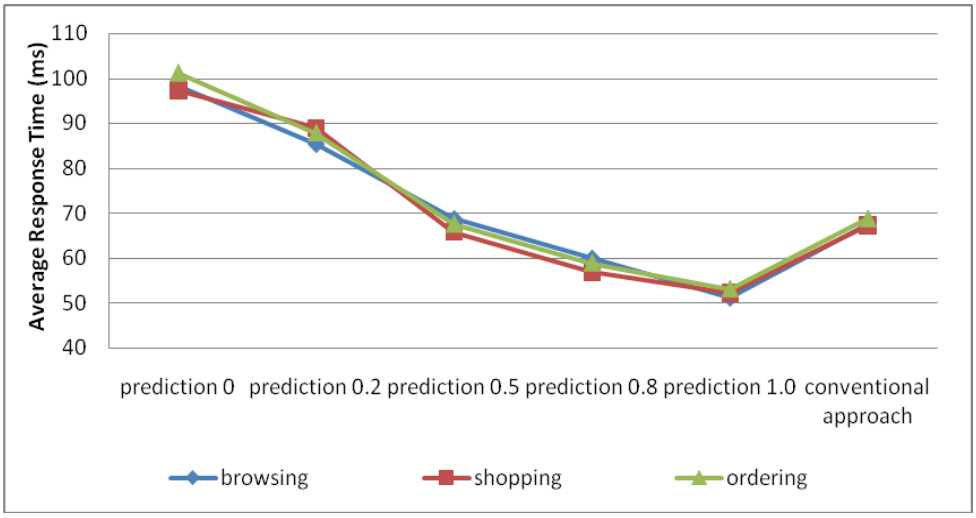


Figure 6-9. Impact of prediction on read operations (IPod)

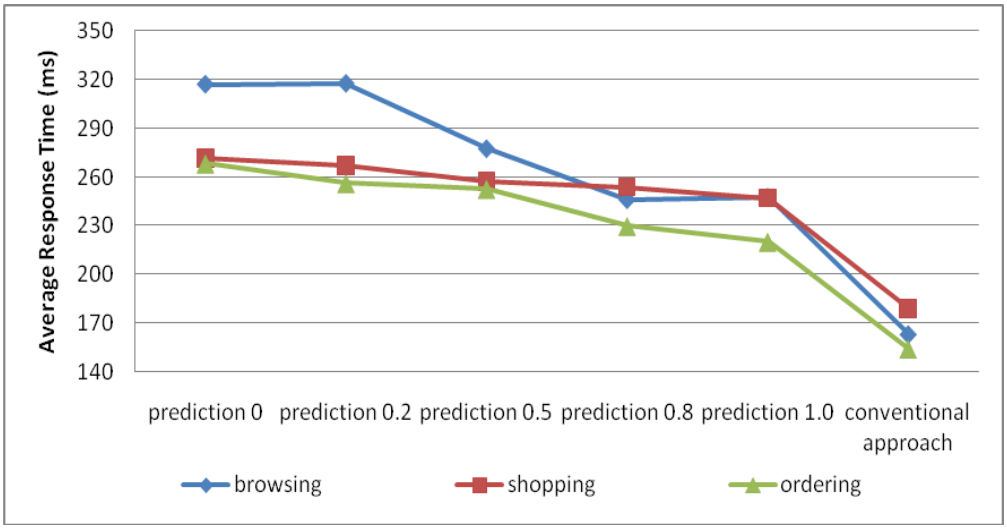


Figure 6-10. Impact of prediction on write operations (IPod)

When looking at the results for write operations, the performance varies in three workloads. As shown in Figure 6-10, the ordering scenario has the best performance among the three workloads in all different prediction settings. Then comes shopping, and finally browsing.

Figure 6-11 presents the impact of prediction accuracy rate on read operations. The result is different from that in the IPod experiment. The thresholds of the prediction accuracy rate for the cooperative approach to beat the conventional approach are different in the three scenarios. Ordering has a relatively lower threshold with a value of 0.2; the overall performance in this scenario is the worst, though.

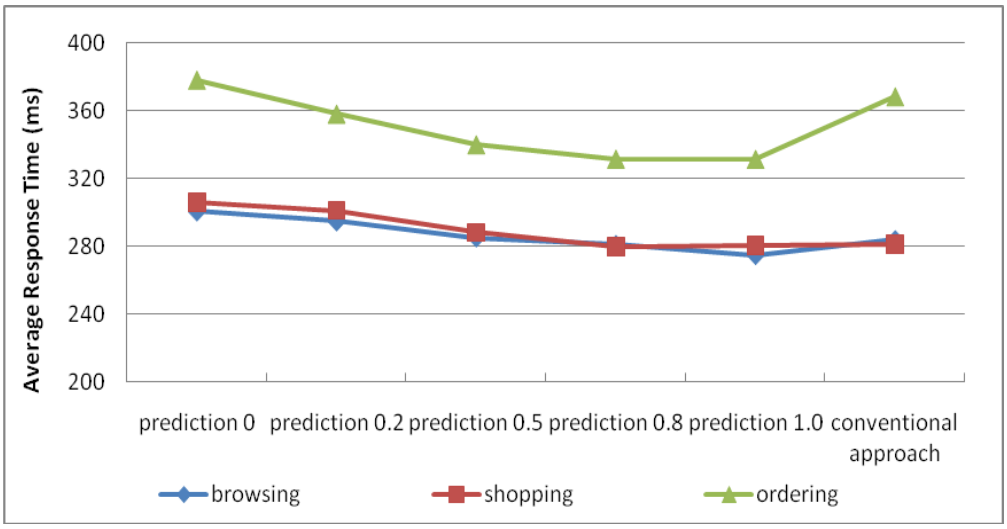


Figure 6-11. Impact of prediction on read operations (G1 Phone)

Figure 6-12 shows the results of write operations. Once again, it's similar to the IPod touch experiments. When the prediction accuracy rate is close to 1, the average response time in the ordering workload is pretty much the same as the one previously shown in Figure 6-8. An interesting finding is that compared to the results of the IPod Touch experiments, the performances of both read and write operations using the G1 phone are poorer with current

experimental setting. For example, the average response time of read operations in the IPod experiments is around 100 milliseconds; however it is around 300 to 400 milliseconds in the G1 Google phone.

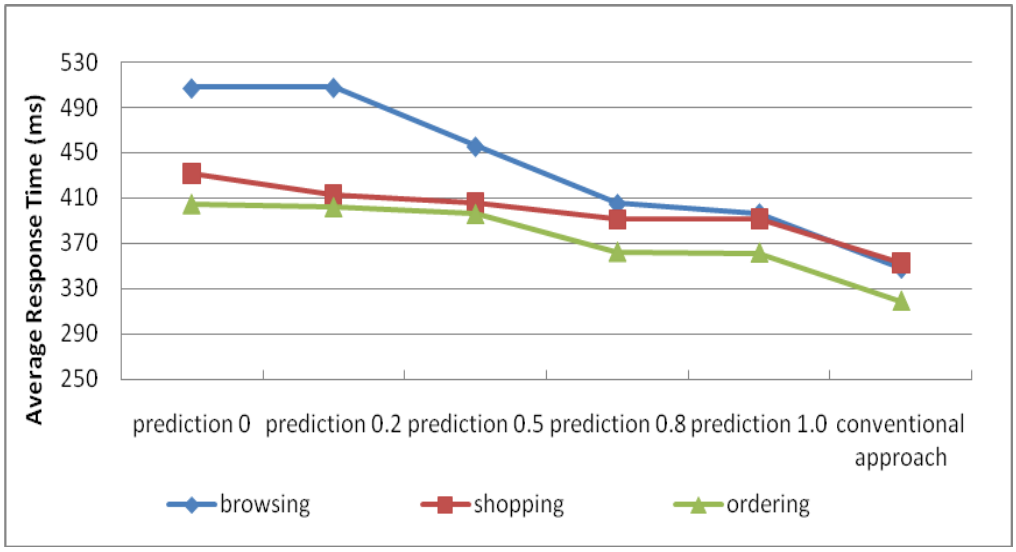


Figure 6-12. Impact of prediction on write operations (G1 Phone)

Impact of Message Size on Performance

In the previous experiments, the size of messages is quite small, around 600 bytes. Experiments in this section aim to investigate the impact of message size on performance. Both the conventional approach and the cooperative approach are used. The G1 Google Phone acts as the client. In order to simulate the real world scenario, two parameters have been added to the end server. They are the processing time and network delay. For the cooperative approach, the prediction accuracy rate is 1, and the percentage of the cached items in the workload is 100%. As a result, the extra server parameters do not have any impact on the cooperative approach for read operations since all the results are directly from the cache.

The network delay is set at 200 milliseconds for settings which have a network delay. To simulate the processing time, the calculation of Fibonacci numbers is used. For settings which have constant processing time, the 37th Fibonacci number is calculated. And for the varying processing time, the 35th Fibonacci number is calculated for the experiments with 50 kb request messages; then the 36th number is used for 100kb experiments, and so forth.

Performance of Read Operations

In these experiments, the workflow is a sequence of 10 GET requests. The time interval is 1 second.

Figure 6-13 shows the experiments on the size of response message. The conventional approach neither has any simulated processing time, nor the simulated network delay. As can be seen from the graph, the average response times in both approaches increase almost linearly as the size of the response message increases. The conventional approach performs slightly better than the cooperative approach in most of the cases.

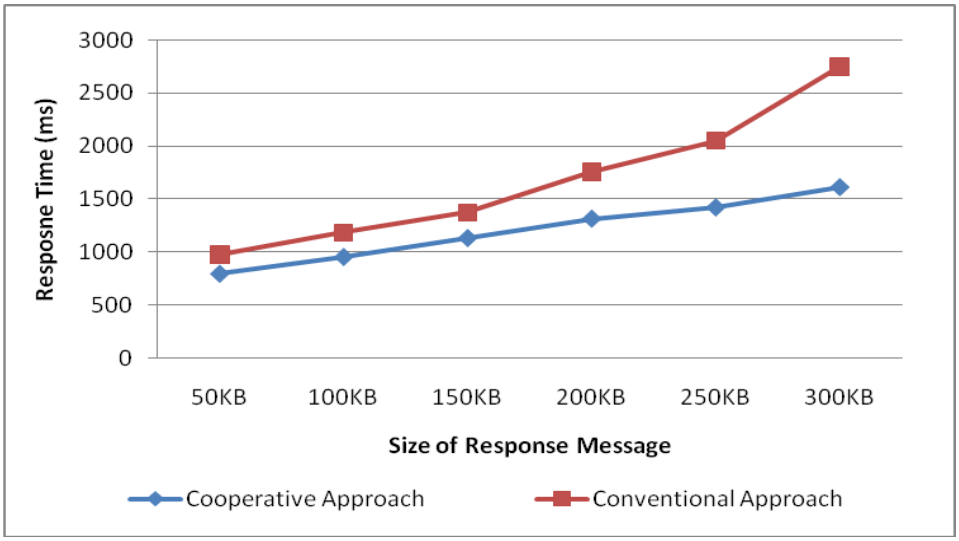


Figure 6-13. Performance with no network delay and no processing time

Figure 6-14 shows the results of the experiments in which the server has additional settings. An extra varying processing time was added. From the graph, with the added extra processing time, the conventional approach performs worse than the cooperative approach when the size of the response message is larger than 150KB. And the response time is no more linearly increasing.

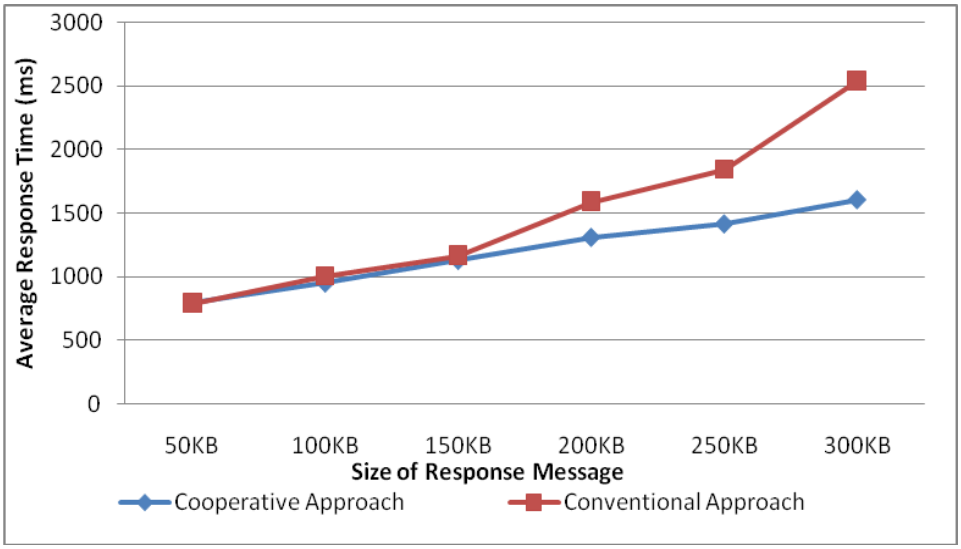


Figure 6-14. Performance with no network delay and varying processing time

Figure 6-15 shows results of experiments with constant extra processing time. This time, the cooperative approach outperforms the conventional approach in all cases.

As can be seen from Figure 6-16, the cooperative approach performs much better than the conventional approach with two parameters added. As before, the response time is not linearly increasing since the calculation of the Fibonacci numbers is not linear.

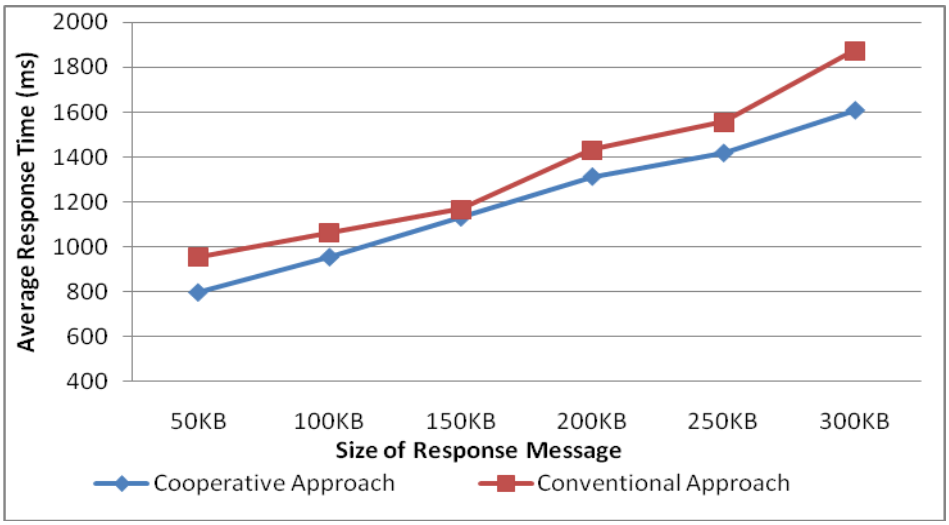


Figure 6-15. Performance with no network delay and constant processing time

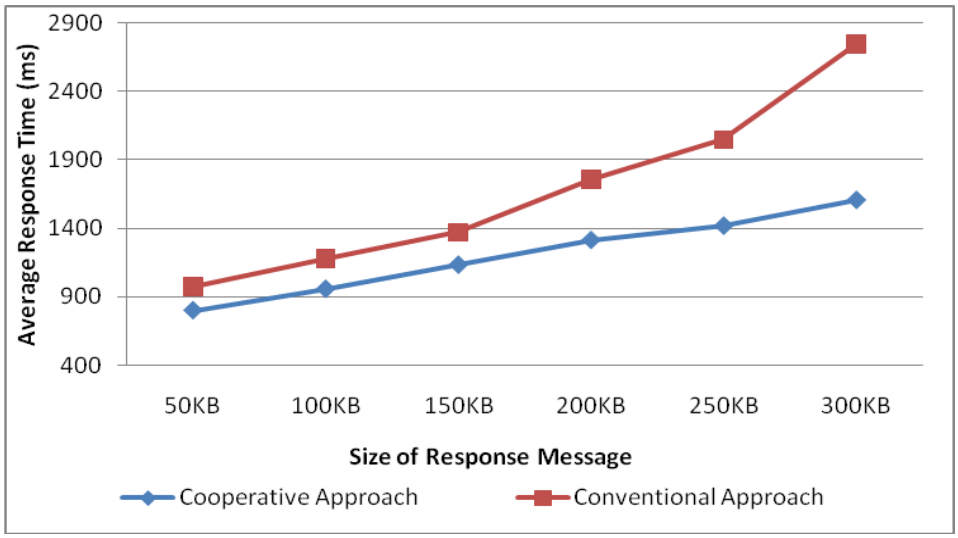


Figure 6-16. Performance with network delay and varying processing time

Figure 6-17 shows the performance of the two approaches with 200 milliseconds delay and a constant processing time. The result is very similar to that of the experiments with only constant processing time. The simulated conventional approach has the same pattern of performance as the one with no parameters added. As the file sizes increases, the average response times in all these approaches increase almost linearly.

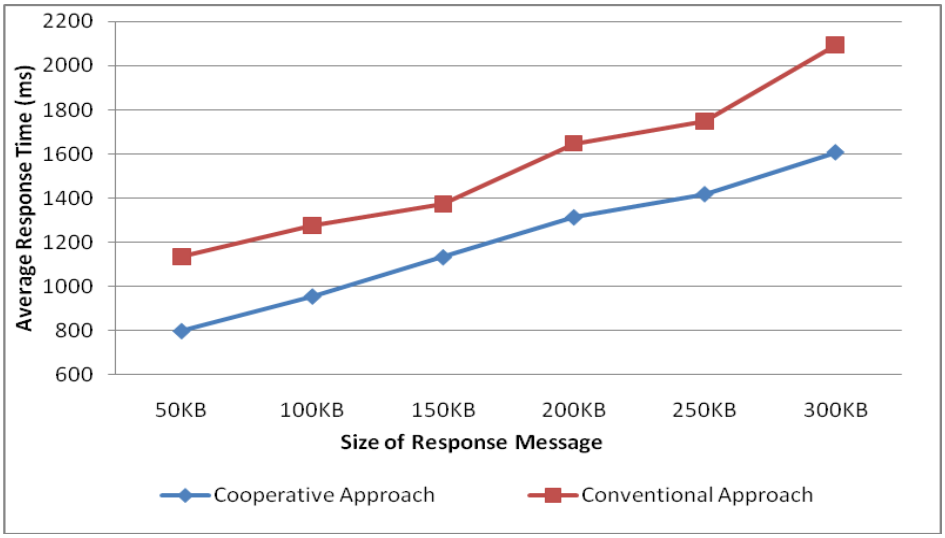


Figure 6-17. Performance with network delay and constant processing time

Performance of Write Operations

These experiments focus on the impact of file size on performance of POST requests. Each experiment uses the same workflow – a sequence of 10 POST requests.

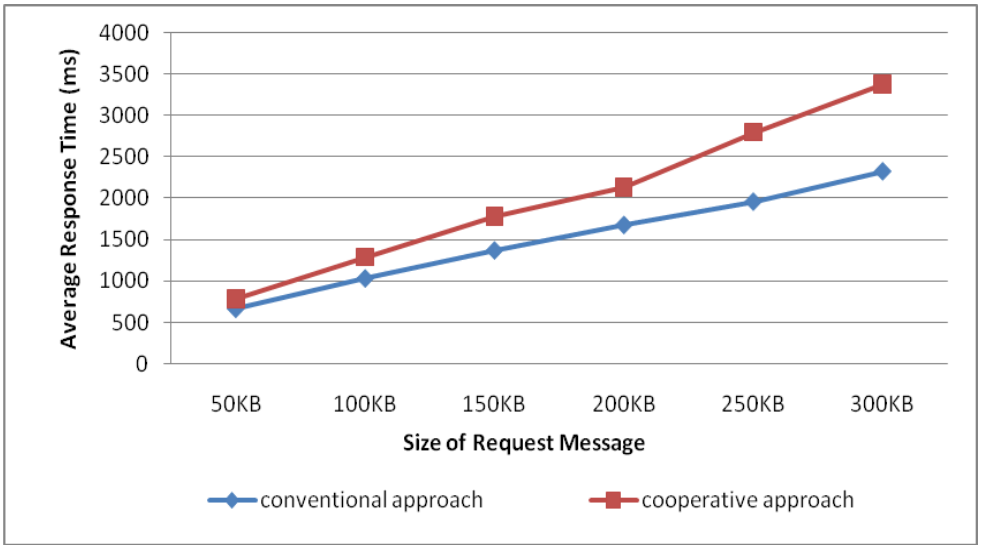


Figure 6-18. Performance with no network delay and no processing time

Figure 6-18 shows the average response times in both approaches. No network delay or processing time is added. As can be seen, the response times in both approaches increase linearly as the size of the request message increases. The cooperative approach performs worse than the conventional approach.

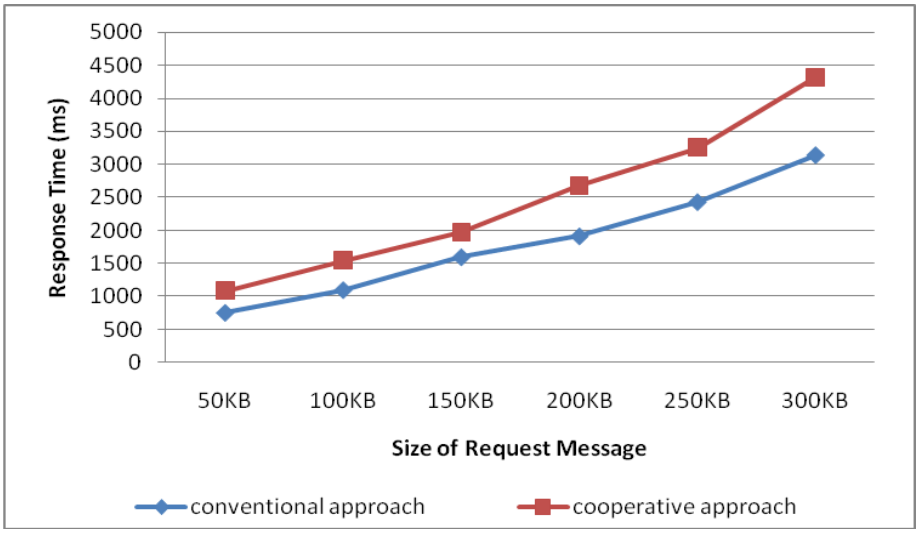


Figure 6-19. Performance with no network delay and varying processing time

Figure 6-19 to Figure 6-22 show the comparison of the two approaches with the same server setting. In all the experiments, the conventional approach outperforms the cooperative approach. This is as expected. This is because when using the cooperative approach to do the POST requests, the requests have to go through the Client-Proxy, then the Server-Proxy, and then finally the server. This actually triples the time it takes to send and receive the data to and from the wire. As a result, for this situation when no resource conflicts exist, the cooperative approach brings no benefits at all.

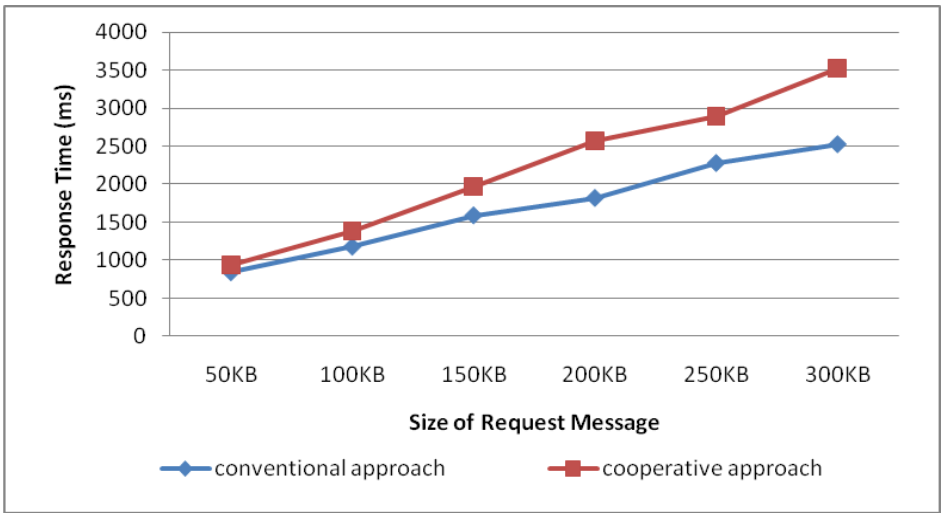


Figure 6-20. Performance with no network delay and constant processing time

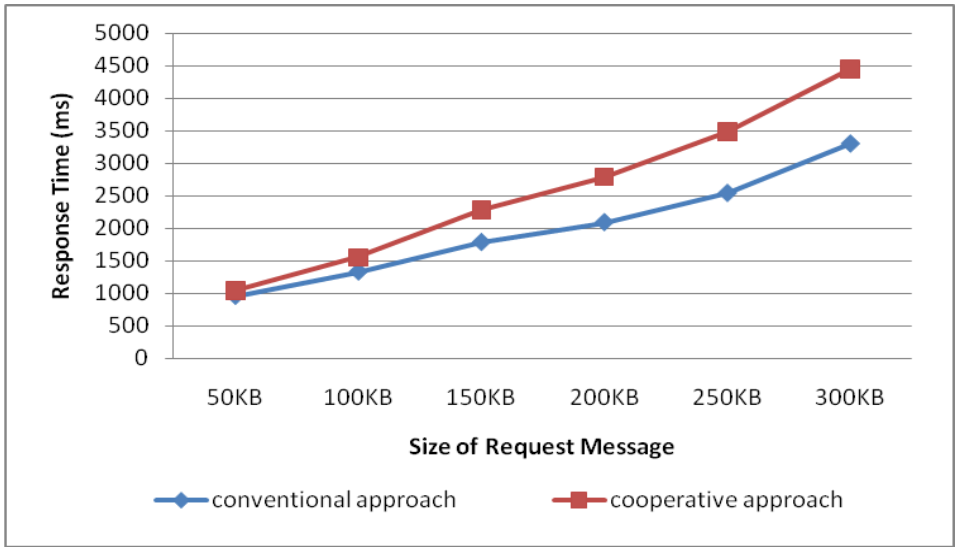


Figure 6-21. Performance with network delay and varying processing time

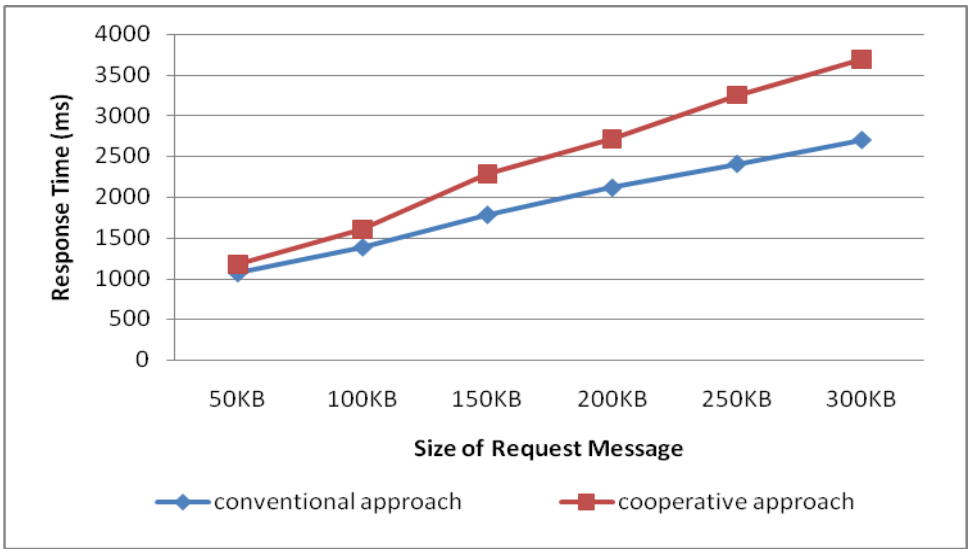


Figure 6-22. Performance with network delay and constant processing time

Impact of Network Delay on Performance

These experiments focus on the impact of network delay on the performance. The request and response message size is 200kb. The network delay varies from 200 milliseconds to 6400 milliseconds. The processing time is constant, using the calculation of the 37th Fibonacci number. The GET workflow consists of 10 consecutive GET requests, 1 second time interval. The POST workflow consists of 10 POST requests, 1 second interval as well.

Figure 6-23 shows the average response times of GET requests in two approaches when network delay increases. As can be seen, the conventional approach performs worse and worse when the network delay increases. However, the performance of the cooperative approach is not affected by the network delay in the Server since it gets results from the Client-Proxy which has already cached the data.

Figure 6-24 shows the average response times of POST requests in two approaches. This time, the network delay has impact on the cooperative approach since all the POST requests have to reach the server. As a result, both approaches show the same pattern: as the simulated

network latency increases, the average response time increases. And the cooperative approach performs slightly worse than the conventional approach.

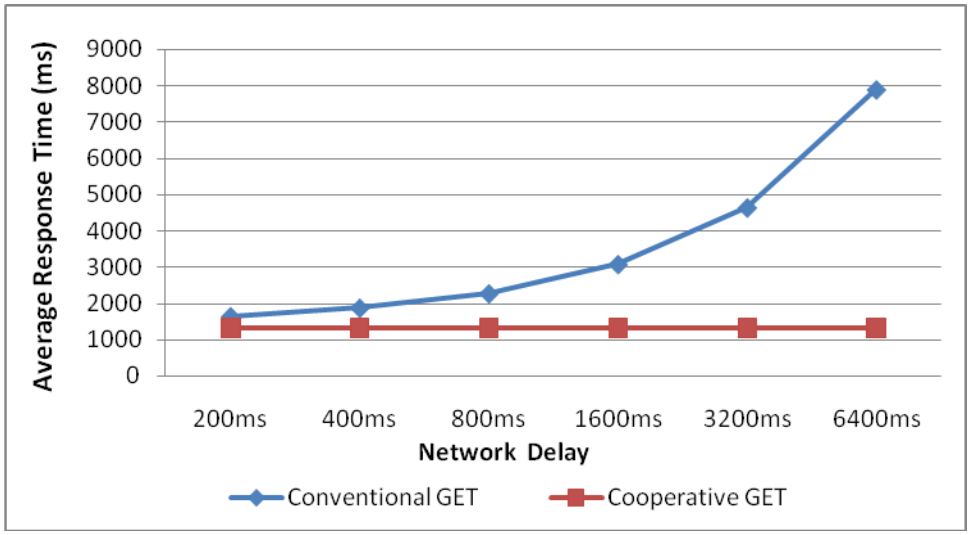


Figure 6-23. Impact of network delay on performance for read operations

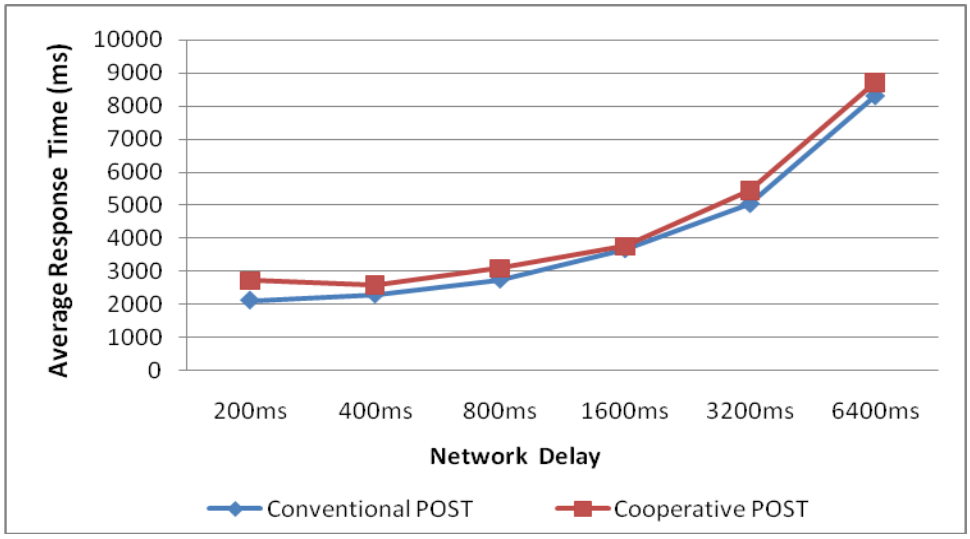


Figure 6-24. Impact of network delay on performance for write operations

Conclusions

The evaluation of the cooperative approach in the mobile experiments is based on the second cooperative scenario: the Open & Centralized Consumer-Provider Cooperation. In this scenario, the consumer (client) shares its workflow with the provider by communicating with the Client-Proxy. The provider prepares locks for resources ahead of time. However, experiments in this chapter aim to examine if the cooperative approach can be applied in mobile computing and how the performance is. Thus the focus is on basic performance parameters and only a single client was used for the tests and resources conflicts were not simulated.

The results of experiments in this chapter further prove the results from previous experiments. For read operations, caching can reduce more overheads of the cooperative approach than the prediction component. On the contrary, prediction can reduce part of the overhead of write operations while caching cannot help at all. Results of the experiments on caching show that the more data is available in the cache, the better the performance is for read operations. As for the experiments on prediction accuracy rate, performance of both types of operations improves as the prediction accuracy rate increases. When the size of request and response message increases, the performance decreases. The cooperative approach has better performance for read operations compared to the conventional approach when the processing time and network delay are added to the end server. While considering the network delay, the cooperative approach show great improvement on performance for read operations as the network delay increases. Performance for write operations in both approaches has very similar pattern when the network delay increases. But still, the conventional approach is slightly better.

In general, the performance for read operations using the cooperative approach is improved compared to the conventional approach. However, results show that the performance for write operations using the conventional approach is better. This is no surprise. In the mobile

experiments, only the single-client workload is simulated for the evaluation since this evaluation only aims at the performance in basic scenarios. As for the single-client workload, there are no resource conflicts happening at the server side since no other clients compete for the resources with the only client simulated. Therefore, the pre-requesting and locking in the cooperative approach are of little use and bring overheads rather than benefits to this single client. Due to the un-cacheable nature of write operations, the overheads are much more obvious than those in read operations.

However, performance will be greatly improved if multiple clients are consuming services simultaneously. This has been proved by experimental results previously shown in last chapter. As shown in Figure 5-6, Figure 5-11, and Figure 5-13, there is significant improvement in performance using multi-client scenarios.

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

Conclusions

Cooperation has been highly emphasized in various areas. However, cooperation among service consumers and service providers has not yet been discussed in WS. This research presents a cooperative approach to improve experiences of WS consumers. Two key aspects of the experiences of consumers are: if the consumer can get the service; how fast can the consumer get the service. As a result, the performance and service availability are two important factors in the goals of the cooperation. Based on the extent of the cooperation, three scenarios are proposed. The Closed Consumer-Provider cooperation presents the basic cooperation. In this scenario, there is only one communication channel for a consumer to consume services; that is to communicate and cooperate with the provider directly. The provider then makes all the decisions about who gets what resources based on the information provided by all the consumers. The second scenario extends the first one and grants permission to consumers to negotiate resources themselves. However, this negotiation is indirect that the provider acts as the agent. The third one - the Open & Decentralized Consumer-Provider cooperation creates an open environment for the consumers to cooperate directly. The provider is no more the decision maker but the executor who carries out the consumers' decisions on the resources.

In this research, two proxies are introduced to implement the cooperation. The Server-Proxy helps to release the server from heavy resource coordination. The Client-Proxy helps the service consumers to cooperate with the service provider by sharing (part of) the consumer's workflow for pre-processing. A re-negotiation mechanism and a reservation-based locking protocol are proposed to help consumers to adapt to changes at runtime. This further enhances the experience of service consumers by providing them with choices and decisions rather than informing them

that services are unavailable. A caching component and a simple prediction model are proposed to help improve the performance for the client.

The cooperative approach has been evaluated with experiments using the first and second cooperation scenario. Results from those experiments show that:

- The overheads caused by the cooperative approach are relatively large compared to the conventional approach. However, with proper caching percentage and prediction accuracy rate, the improvement in performance using the cooperative approach is obvious.
- Caching helps a lot for the improvement in performance for read operations. Prediction can bring benefits for both operations but the improvement is greater in the write operations than in the read operations.
- The cooperative approach is designed for the situation in which consumers compete for resources on the server. The experiments with multi-client workloads prove that when consumers compete for the resources, the cooperative approach shows great improvement in performance compared to the conventional approach. The thresholds for the prediction accuracy rate and the caching percentage are also lower compared to those in experiments with single-client workloads.
- Results from the same experiments with mobile devices support the above conclusions. Since only the single-client workload was used in the mobile experiments, the performance of write operations using the conventional approach outperforms that using the cooperative approach. This has been explained in the conclusions of previous chapter.
- Performance can be affected by the size of the request and response messages. The larger the size is, the greater the latency is. With simulated processing time and network latency, the cooperative approach out-performs the conventional one for read operations. However, it's quite the contrary for write operations since only the single-client workload is used in the experiments with mobile devices and no resource conflicts exist.
- Simulated network latency can greatly affect the performance of the conventional approach. For read operations, the cooperative approach enables improvement in performance with the help of the cache and prediction. This brings great benefits to service clients with poor or unstable network connections, especially for mobile clients. For write operations, the conventional approach is better.

The contributions of this research include the following:

- Introducing the idea of cooperation in WS to improve the experience of service consumers
- Proposing three possible cooperative scenarios for different business and network environments

- Evaluating the proposed cooperative approach with experiments
- Finding out the situations when the cooperative approach has the most significant improvement

Future Work

In the current implementation and experiments, only the first two cooperation scenarios have been evaluated. The third cooperation scenario is the most interesting one and needs further research. The future work will focus on the following aspects:

- The prediction model: The prediction model in the current implementation is a very basic one. It all depends on the client's workflow. It cannot handle requests that are not from the pre-set workflows. So the next step is to build a more advanced and fault-tolerance prediction model that can handle unexpected requests and provide compensations for wrong predictions. The model will no longer totally rely on the client's workflow since it's very rare that a client share all of its workflow with a second party. Therefore, the client's behavior should be accurately captured and analyzed.
- The re-negotiation
    - The current re-negotiation is implemented mostly on the Server-Proxy. The next step is to design a re-negotiation component on the Client-Proxy so that the clients can communicate with each other. This will enable the third cooperation scenario: the Open & Decentralized Cooperation.
    - Penalties should be considered in case of dishonest rankings of requests.
    - A more interesting approach for re-negotiation is to let the clients bid for resources if conflicts happen in a situation where clients with lower priorities really need the resource and they are willing to pay more.
- The pre-requesting and the locking: Currently, most of the overheads are caused by the pre-requesting and the locking, especially when the prediction is incorrect. A further step is to find out a more efficient locking policy and a terse pre-requesting protocol to lower the overheads.
- Evaluations: Current evaluation with mobile devices all focuses on the single-client workloads which cannot show the advantages of the cooperative approach designed to work in a competitive environment. Future experiments should be more focused on multi-client workloads. Service availability and some other parameters should also be investigated other than just the average response time in the mobile experiments.

LIST OF REFERENCES

[1] Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F. 2006. Service-Oriented Computing Research Roadmap. Service Oriented Computing (SOC), number 05432 in Dagstruhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik, Schloss Dagstuhl, Germany 2006.

[2] Erradi, A., Padmanabhuni, S., and Varadharajan, N. 2006. Differential QoS support in Web Services Management. In Proceedings of the IEEE international Conference on Web Services, Washington, DC, September 18 - 22, 2006, pp781-788.

[3] Verma, A. and Ghosal, S. 2003. On admission control for profit maximization of networked service providers. In Proceedings of the 12th international Conference on World Wide Web, Budapest, Hungary, May 20 - 24, 2003, pp128-137.

[4] Olshefski, D. and Nieh, J. 2006. Understanding the management of client perceived response time. SIGMETRICS Perform. Eval. Rev. 34, 1 (Jun. 2006), 240-251.

[5] Choi, S. W. Her, J. S. and Kim, S. D. 2007. Modeling QoS Attributes and Metrics for Evaluating Services in SOA Considering Consumers' Perspective as the First Class Requirement. In Proceedings of the the 2nd IEEE Asia-Pacific Service Computing Conference (December 11 - 14, 2007). APSCC. IEEE Computer Society, Washington, DC, 398-405.

[6] Papazoglou, M. P. 2003. Service -Oriented Computing: Concepts, Characteristics and Directions. In Proceedings of the Fourth international Conference on Web information Systems Engineering (December 10 - 12, 2003). WISE. IEEE Computer Society, Washington, DC, 3.

[7] Gottschalk, K., Graham, S., Kreger, H., and Snell, J. 2002. Introduction to Web services architecture. IBM Systems Journal 41, 2, (2002).

[8] Lee, K.C., Jeon, J. H., Lee, W.S., Jeong, S.H., and Park ,S.W., 2003. QoS for Web services: Requirements and Possible Approaches. W3C Working Group Note 25 (Nov. 2003)

[9] Abu-Ghazaleh, N., Lewis, M. J., and Govindaraju, M. 2004. Differential Serialization for Optimized SOAP Performance. In Proceedings of the 13th IEEE international Symposium on High Pezzformance Distributed Computing (June 04 - 06, 2004). High Performance Distributed Computing. IEEE Computer Society, Washington, DC, 55-64.

[10] Suzumura, T., Takase, T., and Tatsubori, M. 2005. Optimizing Web Services Performance by Differential Deserialization. In Proceedings of the IEEE international Conference on Web Services (July 11 - 15, 2005). ICWS. IEEE Computer Society, Washington, DC, 185-192.

[11] Abdelzaher, T. F., Shin, K. G., and Bhatti, N. 2002. Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. IEEE Trans. Parallel Distrib. Syst. 13, 1 (Jan. 2002), 80-96.

[12] Chiu, K., Govindaraju, M., and Bramley, R. 2002. Investigating the Limits of SOAP Performance for Scientific Computing. In Proceedings of the 11th IEEE international Symposium on High Performance Distributed Computing (July 24 - 26, 2002). High Performance Distributed Computing. IEEE Computer Society, Washington, DC, 246.

[13] Werner, C., Buschmann, C., and Fischer, S. 2004. Compressing SOAP Messages by using Differential Encoding. In Proceedings of the IEEE international Conference on Web Services (June 06 - 09, 2004). ICWS. IEEE Computer Society, Washington, DC, 540.

[14] Elnikety, S., Nahum, E., Tracey, J., and Zwaenepoel, W. 2004. A method for transparent admission control and request scheduling in e-commerce web sites. In Proceedings of the 13th international Conference on World Wide Web (New York, NY, USA, May 17 - 20, 2004). WWW '04. ACM, New York, NY, 276-286.

[15] Verma, A. and Ghosal, S. 2003. On admission control for profit maximization of networked service providers. In Proceedings of the 12th international Conference on World Wide Web (Budapest, Hungary, May 20 - 24, 2003). WWW '03. ACM, New York, NY, 128-137.

[16] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S. 2002. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing 6, 2 (Mar. 2002), 86-93.

[17] Fielding, R. T. 2000 Architectural Styles and the Design of Network-Based Software Architectures. Doctoral Thesis. UMI Order Number: AAI9980887, University of California, Irvine.

[18] Devaram, K. and Andresen, D. 2003. SOAP Optimization via Client-side Caching. In Proceedings of the International Conference on Web Services (ICWS '03, Las Vegas, Nevada, USA, June 23 - 26, 2003).

[19] Cerami C. 2002. Chapter 6: WSDL Essentials. In Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. O'Reilly Media, (Feb. 2002).

[20] Kalepu, S., Krishnaswamy, S., and Loke, S.W. 2003. Verity: a QoS metric for selecting Web services and providers. In proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03,). pp. 131-139

[21] Maximilien, E.M. and Singh, M.P., 2002. Reputation and endorsement for web services. ACM SIGEcom Exchanges, 3(1):24–31, ACM Special Interest Group on E-Commerce.

[22] Ludwig, H. Keller, A., Dan, A., King, R.P., and Franck, R. 2002. Web Service Level Agreement (WSLA) Lnnguoge Specifcarion. IBM Corporation. November 2002.

[23] Rajesh, S. and Arulazi, D. Quality of Service for Web Services—Demystification, Limitations, and Best Practices. http://www.developer.com/services/article.php/2027911

[24] Mani, A. and Naqarajan, A. 2002. Understanding quality of service for Web services. http://www-128.ibm.com/developerworks/library/ws-quality.html

[25] Liu, X. 2007. Model-driven Dual Caching for Nomadic Service-Orientied Archtecture Clients, University of Saskatchewan, Saskatoon, Saskatchewan, CA [Master Thesis].

[26] Liu, X. and Deters, R. 2007. An efficient dual caching strategy for web service-enabled PDAs. In Proceedings of the 2007 ACM Symposium on Applied Computing (Seoul, Korea, March 11 - 15, 2007). SAC '07. ACM, New York, NY, 788-794.

[27] Friedman, R. 2002. Caching web services in mobile ad-hoc networks: opportunities and challenges. In Proceedings of the Second ACM international Workshop on Principles of Mobile Computing (Toulouse, France, October 30 - 31, 2002). POMC '02. ACM, New York, NY, 90-96.

[28] Takase, T. and Tatsubori, M. 2004. Efficient Web Services Response Caching by Selecting Optimal Data Representation. In Proceedings of the 24th international Conference on Distributed Computing Systems (Icdcs'04) (March 24 - 26, 2004). ICDCS. IEEE Computer Society, Washington, DC, 188-197.

[29] Wang, J. 1999. A survey of web caching schemes for the Internet. SIGCOMM Comput. Commun. Rev. 29, 5 (Oct. 1999), 36-46.

[30] Connolly, T. M., and Begg, C. E. 2002. Database Systems: A Practical Approach to Design, Implementation, and Management. Third Edition, Pearson Education Limited, Harlow, England.

[31] Mock, M., Gergeleit, M., and Nett, E. 1997. Cooperative Concurrency Control on the Web. In Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '97) (October 29 - 31, 1997). FTDCS. IEEE Computer Society, Washington, DC, 118.

[32] Zhao, W., Moser, L. E., and Melliar-Smith, P. M. 2005. A Reservation-Based Coordination Protocol for Web Services. In Proceedings of the IEEE international Conference on Web Services (July 11 - 15, 2005). ICWS. IEEE Computer Society, Washington, DC, 49-56.

[33] Alonso, G., Agrawal, D., Abbadi, A. E., Kamath, M., Günthör, R., and Mohan, C. 1996. Advanced Transaction Models in Workflow Contexts. In Proceedings of the Twelfth international Conference on Data Engineering (February 26 - March 01, 1996). S. Y. Su, Ed. ICDE. IEEE Computer Society, Washington, DC, 574-581.

[34] Hollingsworth, D. 1994. Workflow Management Coalition The Workflow Reference Model. Workflow Management Coalition, Document Number TC00-1003.

[35] Huang, H. and Mason, R. A. 2006. Model Checking Technologies for Web Services. In Proceedings of the the Fourth IEEE Workshop on Software Technologies For Future Embedded and Ubiquitous Systems, and the Second international Workshop on Collaborative Computing, integration, and Assurance (Seus-Wccia'06) - Volume 00 (April 27 - 28, 2006). SEUS-WCCIA. IEEE Computer Society, Washington, DC, 217-224.

[36] Havey, M. 2005. Essential Business Process Modelling. First Edition O'Reilly, 2005, pp. 350.

[37] Windows Workflow Foundation (WF). Micorsoft, http://msdn.microsoft.com/en-us/netframework/aa663328.aspx.

[38] State Machine Workflows. MSDN library from microsoft. http://msdn.microsoft.com/en-us/library/ms735945(VS.85).aspx.

[39] TPC-transaction processiong performance council. [Online]. 2007(06/21), Available: http://www.tpc.org/.

[40] Karagiannis, T., Molle, M., Faloutsos M. and Broid, A. 2004. A nonstationary poisson view of internet traffic . IEEE Infocom 2004-Confrence on Computer Communications - Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, 2004, pp. 1558-1569.

[41] Dyachuk, D. and Deters, R. Improving performance of Composite Web Services. 2007. In Proceedings of IEEE International Conference on Service-Oriented Computing and Applications, pages 147 – 154, June 19-20, 2007

[42] Pallis, G., Vakali, A., and Pokorny, J. 2007, A Clustering-based Prefetching Scheme on a Web Cache Environment, Comput Electr Eng (2007), doi:10.1016/j.compeleceng.2007.04.002

[43] Erradi, A. and Maheshwari, P. 2005. wsBus: QoS-Aware Middleware for Reliable Web Services Interactions. In Proceedings of the 2005 IEEE international Conference on E-Technology, E-Commerce and E-Service (Eee'05) on E-Technology, E-Commerce and E-Service (March 29 - April 01, 2005). EEE. IEEE Computer Society, Washington, DC, 634-639. DOI= http://dx.doi.org/10.1109/EEE.2005.148

[44] Rosu, M.-C 2007. A-SOAP: Adaptive SOAP Messaage Proccesing and Compression. In Proceedings of the IEEE International Conference on Web Services. ICWS 2007.

[45] Conti, M., Kumar, M., Das, S. K., and Shirazi, B. A. 2002. Quality of Service Issues in Internet Web Services. IEEE Transactions on Computers, vol. 51, no. 6, pp. 593-594, June, 2002.

[46] Teng, W. and Chang, C. 2005. Integrating Web Caching and Web Prefetching in Client-Side Proxies. IEEE Trans. Parallel Distrib. Syst. 16, 5 (May. 2005), 444-455. DOI= http://dx.doi.org/10.1109/TPDS.2005.56

[47] Yin, L. and Cao, G. 2006. Supporting Cooperative Caching in Ad Hoc Networks. IEEE Transactions on Mobile Computing 5, 1 (Jan. 2006), 77-89. DOI= http://dx.doi.org/10.1109/TMC.2006.15

[48] Alonso, G., Casati, F., Kuno, H., and Machiraju, V. Web Services. Springer Verlag, 2004

[49] Pautasso, C., Zimmermann, O., and Leymann, F. 2008. Restful web services vs. "big"' web services: making the right architectural decision. In Proceeding of the 17th international Conference on World Wide Web (Beijing, China, April 21 - 25, 2008). WWW '08. ACM, New York, NY, 805-814. DOI= http://doi.acm.org/10.1145/1367497.1367606

[50] Schäfer, M., Dolog, P., and Nejdl, W. 2008. An environment for flexible advanced compensations of Web service transactions. ACM Trans. Web 2, 2 (Apr. 2008), 1-36. DOI= http://doi.acm.org/10.1145/1346237.1346242

[51] Monaco, F. J., Nery, M., and Peixoto, M. M. 2009. An orthogonal real-time scheduling architecture for responsiveness QoS requirements in SOA environments. In Proceedings of the 2009 ACM Symposium on Applied Computing (Honolulu, Hawaii). SAC '09. ACM, New York, NY, 1990-1995. DOI= http://doi.acm.org/10.1145/1529282.1529724

[52] Diamadopoulou, V., Makris, C., Panagis, Y., and Sakkopoulos, E. 2008. Techniques to support Web Service selection and consumption with QoS characteristics. J. Netw. Comput. Appl. 31, 2 (Apr. 2008), 108-130.

[53] Weiss, A. 2007. Computing in the clouds. netWorker 11, 4 (Dec. 2007), 16-25. DOI= http://doi.acm.org/10.1145/1327512.1327513

[54] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. 25, 6 (Jun. 2009), 599-616.

[55] Liu, S., Wei, J., Ma, Y., and Liu, Y. 2004. Web Service Cooperation Ideology. In Proceedings of the 2004 IEEE/WIC/ACM international Conference on Web intelligence (September 20 - 24, 2004). Web Intelligence. IEEE Computer Society, Washington, DC, 537-540. DOI= http://dx.doi.org/10.1109/WI.2004.160 2009), 599-616.