MODEL-DRIVEN DUAL CACHING FOR NOMADIC SERVICE-ORIENTED-ARCHITECTURE CLIENTS

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

Xin Liu

ABSTRACT

Mobile devices have evolved over the years from resource constrained devices that supported only the most basic tasks to powerful handheld computing devices. However, the most significant step in the evolution of mobile devices was the introduction of wireless connectivity which enabled them to host applications that require internet connectivity such as email, web browsers and maybe most importantly smart/rich clients. Being able to host smart clients allows the users of mobile devices to seamlessly access the Information Technology (IT) resources of their organizations.

One increasingly popular way of enabling access to IT resources is by using Web Services (WS). This trend has been aided by the rapid availability of WS packages/tools, most notably the efforts of the Apache group and Integrated Development Environment (IDE) vendors. But the widespread use of WS raises questions for users of mobile devices such as laptops or PDAs; how and if they can participate in WS. Unlike their "wired" counterparts (desktop computers and servers) they rely on a wireless network that is characterized by low bandwidth and unreliable connectivity.

The aim of this thesis is to enable mobile devices to host Web Services consumers. It introduces a Model-Driven Dual Caching (MDDC) approach to overcome problems arising from temporarily loss of connectivity and fluctuations in bandwidth.

ACKNOWLEDGMENTS

First of all, I would like to sincerely thank my supervisor, Dr. Ralph Deters, for his persistent support, guidance, help, and encourage during the whole process of my study and my thesis.

I would like to thank the members of my advisory committee: Dr. Julita Vassileva, Dr. John Cooke, and Dr. Chris Zhang (external) for their valuable suggestion and comments.

I would like to thank the students, staff and faculty of the Computer Science Department and especially to students of MADMUC lab for their support. Also sincerely thanks to Ms. Jan Thompson for her help and support.

I would like to thank my family for their support and love all the time and dedicate my thesis to them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ASP	Active Server Pages
B2B	Business to Business
BPEL	Business Process Executable Language
BPM	Business Process Management
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CSC	Client Side Cache
DAML+OIL	DARPA Agent Markup Language + Ontology Inference Layer
DAML-S	DAML-based Web service ontology
DL	Description Logic
GD-size	GreedyDaul-size
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
IT	Information Technology
JMI	Java Metadata Interface
JSP	JavaSever Pages
LBS	Location-Based Service
LFU	Least Frequently Used
LRU	Least Recently Used
MDDC	Model Driven Dual Caching
ORB	Object Request Broker
OS	Operating System
OWL	Ontology Web Language
OWL-S	Ontology Web Language for Services
P2P	Peer to Peer
PSC	Provider Side Cache
PDA	Portable Digital Assistant
RDF	Resource Description Framework
RDFS	RDF Schema
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SO	Service Oriented
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TTL	Time-To-Live
UDDI	Universal Description and Discovery Interface
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
Wi-Fi	Wireless Fidelity
WS	Web Services
WSCI	Web Services Choreography Interface
WS-CDL	Web Services Choreography Description Language

WSDL	Web Service Description Language
WSFL	Web Services Flow Language
WWW	World Wide Web
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language

CHAPTER 1

INTRODUCTION

Apple's Newton (1992 - 98), the first commercially available Personal Digital Assistant (PDA), was a resource and connectivity constrained device (8 MB RAM, infrared connection, serial Port) designed to support users with simple data processing/collecting tasks. While being a commercial failure, the Newton succeeded in popularizing the idea of the PDA and created a new type of computational devices.

The ongoing evolution of PDAs has led to resource rich devices such as the IPaq hx2790 (256 MB, PXA70 624 MHz, 240 x 320 display with 65,536 colors) that offers computational resources comparable to previous desktops a few years earlier. However, the most significant improvement for PDAs is not memory or CPU. It is the standardization of the software platform (Palm-OS, Microsoft-Windows Pocket-PC, and Linux) and the integration of 802.11b/g and Bluetooth. The introduction of standard Operating Systems (OS) and standardized wireless communication has enabled the PDA to evolve beyond a simple data processing/collecting device requiring frequent synchronization with the desktop. Wireless connection to a network enables users to use their PDAs in much the same way as using their desktops. Besides calendar and address book features, PDAs run web browsers, email clients, and (thanks to the standardization of the OS) smart clients for legacy business applications. Modern PDAs empower users to perform complex tasks (business processes) and support a spatial decoupling of users from their main computational resources.

Approximately at the same time the spatial decoupling of users from resources gained momentum, efforts to decouple IT components lead to the concept of Service-Orientation (SO [1]). SO is a design & integration paradigm based on the notion of well-defined, loosely coupled services. Within SO, services are viewed as computational elements that expose functionality in a platform independent manner and can be described, published, discovered, orchestrated and consumed across language, platform and organization boundaries. The Service-Oriented Architecture (SOA [2]), first introduced by Gartner in 1996 [3], is a conceptual framework that identifies the service-consumer, the service-provider, and the service registry as its basic architectural elements. Figure 1-1 shows the topology of SOA.

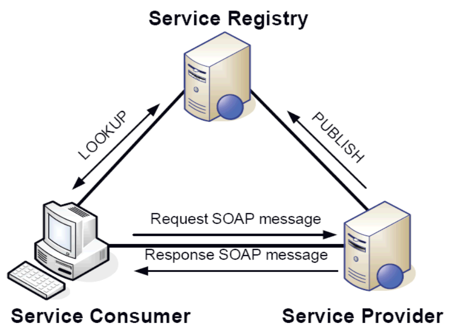


Figure 1-1. Topology of Service-Oriented Architectures

Application components deployed on a system (e.g. a legacy system) can be transformed into network-based services, enabling organizations to reuse them rather than to develop new ones. SOA can be achieved using different technologies, but Web Services (WS [4]) is the most commonly used approach due to the standardization efforts and the available tools/infrastructure. WS were designed to ease the interoperability of services by utilizing the web and open eXtensible Markup Language (XML [5]) based standards. Simple Object Access Protocol (SOAP [6]), Web Service Description

Language (WSDL [7]) and Universal Description and Discovery Interface (UDDI [8]) are the three key components of WS, which provide specifications for the service communication protocol, service description, and service discovery, respectively. The interaction between the service consumer and service provider is achieved by exchanging SOAP messages, which rely on existing protocols such as HyperText Transfer Protocol (HTTP [9]), or Simple Mail Transfer Protocol (SMTP [10]) as the transport mechanism. SOAP messages are XML documents that contain all the relevant information for issuing a request or returning the results of a request in their header and body.

WS enjoy wide support due to the acceptance of SO and the availability of WS packages/tools. Using IDE tools and other software packages, it is easy for programmers to expose application interfaces and/or consume existing interfaces. As the XML-based WS begin to emerge as the "lingua franca" for the development of loosely coupled distributed systems they become the preferred way for building large, distributed, heterogeneous systems. The major implication of loose-coupling is that components (e.g. legacy systems) can be easily connected and work together. By retrieving the service description from the provider, it is possible to *automatically* generate the stubs needed to communicate with the services. Once services are deployed, it is simple to extend these services to other consumer devices such as PDAs. This thesis defines a nomadic client in SOA/WS as an application residing on a mobile device (e.g. laptop, PDA, Smartphone) and consuming services in SOA/WS through a wireless network.

However, the "standard" WS scenario assumes service providers and consumers are static and always connected, creating an obstacle for laptops and PDAs to participate. Unlike their "wired" counterparts, mobile devices rely on wireless networks that are characterized by low bandwidth, unreliable connectivity, and expensive usage charges [11]. In a wireless environment, voluntary disconnections (e.g. to save battery power) and involuntary ones (e.g. no signal coverage) are fairly common. When the client is disconnected, it cannot access services, and applications do not function properly.

To overcome wireless connectivity problems, it is necessary to use caching and prefetching techniques. Since caches have long been used to handle disconnections for nomadic clients [12] they seem to be a good approach for supporting nomadic WS consumers. WS tend to use the HTTP protocol to exchange SOAP messages [13], making the interactions between the service consumer and the service provider remarkably similar to "normal" web client-server interactions. By intercepting and analyzing SOAP messages, it becomes possible to replicate and reuse this information within a caching system.

This thesis focuses on the issues of how to use *models*, which include the service model, the client model and the wireless network model, to guide caching and prefetching in the context of nomadic SOA/WS participants. In order to support seamless access and overcome short periods of disconnection, this thesis introduces the idea of Model-Driven Dual Caching (MDDC) and evaluates its effectiveness under various conditions.

This thesis is structured as follows: Chapter two gives a description of the problems of caching services in SOA/WS. Chapter three is a survey of literature based on issues of caching services. Chapter four presents the architecture and functionalities of MDDC. Chapter five and Chapter six present the evaluation results of MDDC. Chapter seven presents the conclusions and future work.

CHAPTER 2

PROBLEM DEFINITION

Mobile devices are increasingly often seen as legitimate client platforms in distributed systems. The advances of wireless technologies and the improvement regarding the form factor and computational resources are key factors in this new trend.

2.1 Scenario

Figure 2-1 shows an abstract scenario of consuming services via a wireless network. Typically, a service-consuming scenario includes three parts: service clients, service providers, and the wireless network.

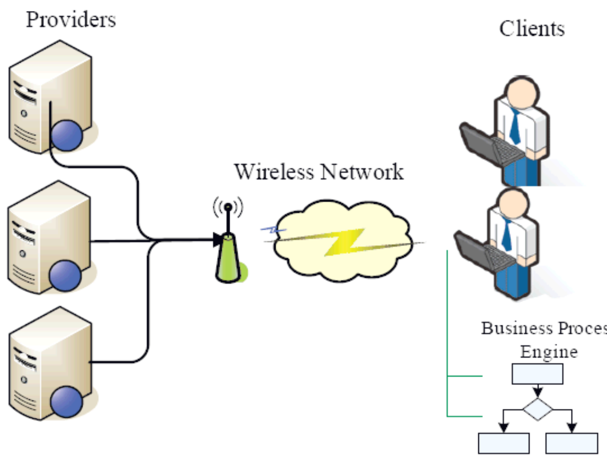


Figure 2-1. A scenario of service consuming with mobile devices

Service Clients: service clients are the consumers of services. In this scenario, clients are consuming services with laptops. They can invoke individual services with simple requests, and/or execute sophisticated business processes that require communication and cooperation of service providers situated in different domains. The small picture below the clients shows a business process specified by Business Process Executable Language (BPEL [14]).

Service Providers: service providers provide services to consumers. Services can be on one homogeneous platform (e.g. only one server is involved), or on heterogeneous platforms that involve multiple servers when the client runs complex business processes. Services perform tasks that vary from fundamental to complex, from answering simple queries to realizing one or more "real world effects", from stand alone to having complex relationships with other services [15]. Service providers describe their services using WSDL. WSDL is a XML-based syntactic description language and does not address the semantics of services. It limits itself to describing a few key aspects of services such as what the service offers, how to invoke it, and where to access it. Service providers may also provide semantics using Ontology Web Language for Services (OWL-S) [16] or other standards.

Network: In this scenario, the communication between the consumer and provider is through a wireless network. Wireless network connectivity can be strong, weak or null [11]. Strong connectivity is the most desirable one, signifying the nomadic client has a high-bandwidth and reliable wireless network connection. Low-bandwidth, intermittent connection, high network latency, or a mix of all results in weak connectivity [11, 17]. When a nomadic client is moving away from wireless access points (e.g. leaving a building with a wireless access point), it is prone to be constrained by weak connectivity that leads to degradation of service quality due to high network latency. Null connectivity can be caused by voluntary disconnections (e.g. to save battery) and involuntary disconnections (due to leaving the range) [11].

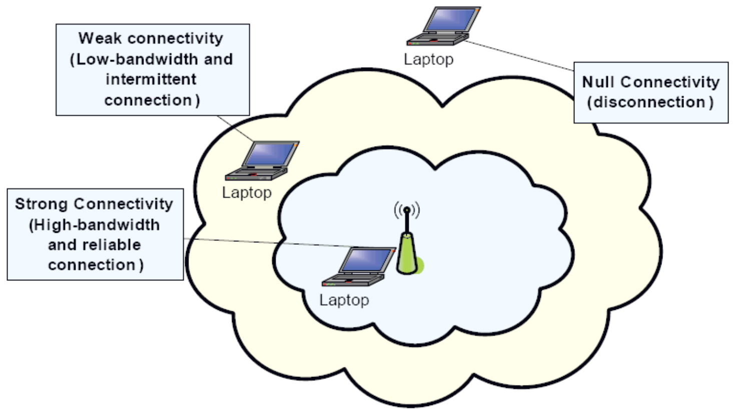
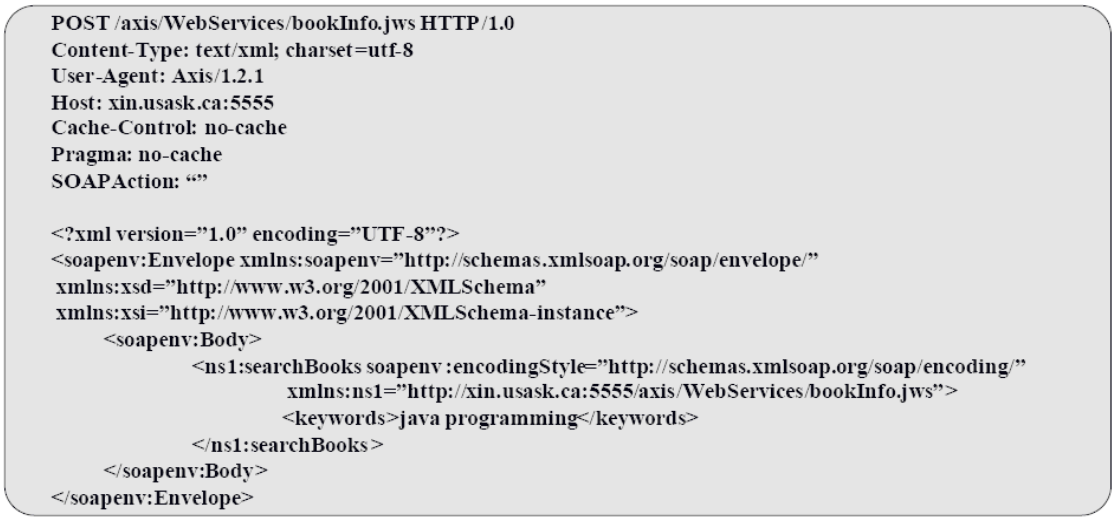

Weak connectivity (Low-bandwidth and intermittent connection)
Laptop
Null Connectivity (disconnection)
Laptop
Strong Connectivity (High-bandwidth and reliable connection)
Laptop

Figure 2-2. Wireless network connectivity

Figure 2-2 shows a possible scenario of wireless connectivity changing: the connectivity is strong when the client is close to a base station; as the client moves far from the station, s/he may suffer from weak connectivity, in which the bandwidth may go down and the signal may be lost from time to time. When the client moves out of the covered area of the station, s/he will lose the connection.

Communication: SOAP defines an XML-based communication protocol for the messages exchanged among WS and it relies on existing transport protocols, such as HTTP and SMTP.

SOA/WS supports both synchronous and asynchronous communication. In synchronous communication, when a service consumer issues a service request, the request will be blocked until a response message is received. In asynchronous communication, after a service request is issued, the consumer forgets it, and performs other actions before the response comes back. SOAP encoding & binding has two styles: Remote Procedure Call (RPC) style and Document-style. RPC-style messages must conform to a structure described by a WSDL file that contains the method name, parameters (request) and return values (response). RPC-style is essentially based on call/response semantics that only supports synchronous calls. Document-style messages contain a single, literal XML document that conforms to a specific XML-Schema, and the application must parse the XML document to find the needed data. Document-style WS support both synchronous and asynchronous calls. Figure 2-3 shows an example of a RPC-style SOAP message over HTTP. Despite designed to be simple and lightweight, SOAP is notorious for the large size required to transmit even little information (e.g. the value of one parameter).



```
POST /axis/WebServices/bookInfo.jws HTTP/1.0
Content-Type: text/xml; charset=utf-8
User-Agent: Axis/1.2.1
Host: xin.usask.ca:5555
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: “”

<?xml version=”1.0” encoding=”UTF-8”?>
<soapenv:Envelope xmlns:soapenv=”http://schemas.xmlsoap.org/soap/envelope/”
 xmlns:xsd=”http://www.w3.org/2001/XMLSchema”
 xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”>
        <soapenv:Body>
                <ns1:searchBooks soapenv:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”
                        xmlns:ns1=”http://xin.usask.ca:5555/axis/WebServices/bookInfo.jws”>
                        <keywords>java programming</keywords>
                </ns1:searchBooks>
        </soapenv:Body>
</soapenv:Envelope>
```

Figure 2-3. A SOAP message over HTTP

2.2 Challenges for Nomadic Clients in SOA/WS

Nomadic computing is an extension of traditional fixed networks [18]. It supports clients to connect to the network by means of wireless network and to roam around from one place to another. “In this model, it is expected that individual organizations will construct their own wireless infrastructures that are linked across organizations by wired internetworks. Yet a user would like to be able to use his or her device even within a foreign organization’s wireless infrastructure.” [11]

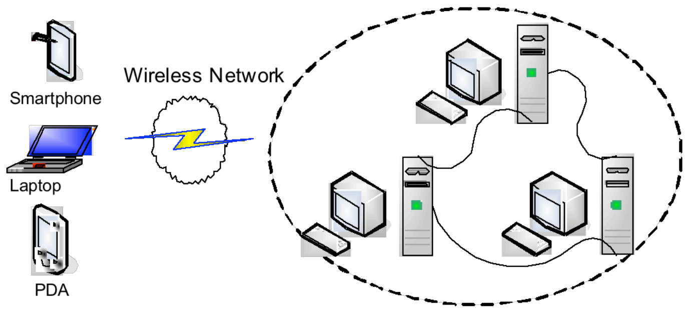


Figure 2-4. Nomadic clients

A nomadic client with a reliable connection appears no different to the service provider than clients using wired devices. They exchange XML/SOAP message as a means of communication. However, as shown in Figure 2-2, one problem with nomadic clients is that the wireless connection can be unstable or unavailable at times. When a consumer is invoking a service with a mobile device, the interruption of request/response message transmission can result in four basic types of interruption scenarios:

Interruption scenario 1: Before the request (SOAP message) is sent, the wireless connection is lost.

Interruption scenario 2: When a SOAP request message is being transmitted to the service, the nomadic client moves to an area with low-bandwidth and unreliable connection. The low bandwidth results in long transmission duration and the transmission may be interrupted by the loss of the connection at anytime.

Interruption scenario 3: The request (SOAP message) has been sent to the service, and the client is waiting for the response (SOAP message). At this time, the connection is lost.

Interruption scenario 4: A similar interruption scenario could also happen when a response SOAP message is being transmitted from the service to the nomadic client.

Synchronous communication is a point-to-point communication, which assumes the service consumer & provider to be connected and running simultaneously. If any interruption scenario happens, the round trip process that includes requesting a service and delivering the service response cannot be completed. The asynchronous communication does not demand the consumer & provider to be connected at the same time. However, the service consumer & provider do not have the ability to detect the condition of the wireless environment before/after sending messages. The four interruption scenarios may cause message losses without being detected.

Caching improves the system availability and scalability by storing information in local memory. The issue here is not so much that the network is wireless, but rather that the system must deliver services in the face of a constantly changing wireless context [11, 12, 18, 19]. It is therefore crucial to use caching technology to mask wireless communication limitations.

2.3 Dual Caching

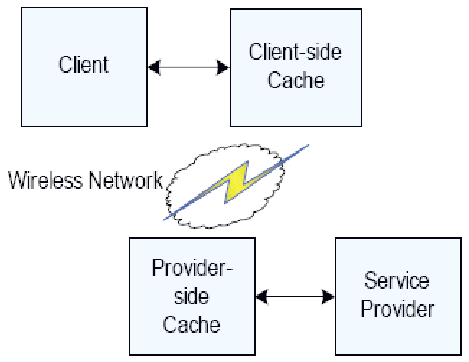


Figure 2-5. Dual-Caching

It is important to note that interruption scenario 1 & 2 can be detected by the client whereas interruption scenario 3 & 4 cannot. Since the loss of connectivity can happen during the sending/receiving of both request messages and response messages, two caches are needed. One cache resides on the client side and the other on the provider side. The Client-Side Cache (CSC) is designed to handle the problems 1 & 2 by queuing requests that could not be sent. It will resend them as soon as the connection is reestablished, and thus create the illusion of seamless connectivity for short-lived disconnectivity. The Provider-Side Cache (PSC) is needed for storing and resending the responses from the provider that have not been successfully transmitted due to the consumer’s loss of connectivity, and thus avoiding confusing the provider.

2.4 Issues in Caching Services for Nomadic Clients

Previous research [20, 21] on Common Object Request Broker Architecture (CORBA) caching, replicated objects and related data on the client-side. The central focus of the previous Object-Oriented technologies, e.g. CORBA and Java Remote Method Invocation (RMI), is the object, which includes data and operations. Interactions between objects are through method signatures. Objects are implemented in a homogenous environment, which makes it possible to move the executable logic to the client side and execute it there. Friedman [22] discusses aspects of caching WS in ad-hoc networks on a more abstract level. The proposed architecture is similar to a P2P system in which some mobile devices are willing to serve as ad-hoc routers that the system can use as proxy caches. However, these approaches are not suitable for WS caching as they consider services as objects and require the ability to cache both data and code. WS focuses on the tasks or functions performed by the services, and services are independent of specific programming languages and operating systems [15, 23]. Services are normally implemented in heterogeneous environments and are not executable when moved to another environment. It is important to have a platform independent solution: to cache the messages exchanged between the service provider and the service consumer.

WS caching differs from Web page caching systems in many aspects. For example, Web page caching only needs to care about the HTTP GET operation as it is for reading pages from the web server. For WS message passing, when SOAP messages are carried by HTTP, the HTTP operation "POST" is used. Thus, it is impossible to identify the cacheability by analyzing HTTP heads. SOAP messages and WSDL specifications have no explicit information about which service can be cached and which cannot, and what the influence of a service is. Therefore, a service model is necessary to tell the caching system which service is cacheable and what its effects (e.g. invalidation) are.

The prefetching/hoarding technique has been widely used to improve the performance of caching systems. In order to prefetch/hoard service requests ahead of time, the caching system must be able to predict the client future behaviors. A client workflow model contains knowledge about the flow of control and data, and thus it can suggest what kind of information and what functionality should be present for the next steps.

As shown in Figure 2-2, mobile devices face dynamic environments. As nomadic clients move from one location to another, the caching system needs to react to the high variability of bandwidth and unexpected connection losses. In order to adapt to these changes, it is necessary to have a model of the wireless network. An ideal model of the environment context should contain a map of the wireless environment that can show the various conditions of the network. For example, when a client is in a car and moving around, a model will enable the caching system to predict the quality of wireless connectivity, e.g. available bandwidth and time to lose the connection.

In conclusion, the more information the caching system has about the services, clients and the wireless network, the better it could perform. This thesis will investigate how models (service, client and wireless network) can be used to enable the caching and prefetching techniques for nomadic clients in SOA/WS, and thus improve access to services.

CHAPTER 3

LITERATURE REVIEW

As pointed out by Field [24], caching improves the performance of network-based applications: "An interesting observation about network-based applications is that the best application performance is obtained by not using the network. This essentially means that the most efficient architecture styles for a network-based application are those that can effectively minimize use of the network when it is possible to do so, through reuse of prior interactions (caching), reduction of the frequency of network interactions in relation to user actions (replicated date and disconnected operation), or by removing the need for some interactions by moving the processing of data closer to the source of the data (mobile code)."

3.1 Standard Issues on Caching

Caching technologies have been studied in many areas of computer science, including web caching, file caching, caching for distributed systems, and caching for mobile computing. Caching has proven to be an effective way for reducing access latency, balancing network traffic, and improving the performance and availability of distributed systems.

3.1.1 Cache Location

In wireless network environments, a client-side cache is necessary for improving the availability of services when clients suffer from disconnection [25, 26] and weak connectivity. Moreover, a client-side cache in local memory allows the client to process requests faster and more efficiently, and thus, reduces the average network latency and bandwidth consumption [27]. The Coda file system [12, 17, 28] was designed for large-scale distributed computing environments. It supports disconnected file Input/Output operations for portable workstations using the local disk as a file cache. Another advantage of the client-side cache is that only the client can have comprehensive and detailed information about itself, such as its access pattern and its execution context.

One motivation for the server-side cache is to offload frequent repeated computations from the server [29]. Moreover, the server-side cache can be responsible for cache policies. For example, the server of the Coda file system generated a callback when a client caches a file or directory to maintain cache consistency. This method reduces server-client interactions while maintaining strong consistency. A proxy cache can be in a stationary server close to the client, e.g. a base station, which has more computation resources and is connected to the server via wired links. The proxy cache could also act as an intermediary traffic balancer by forwarding requests to the least utilized server [27]. Friedman [22] suggested "a hierarchy of proxy caches within the ad-hoc network for popular services in order to further reduce the traffic inside wireless networks". ARTour Web Express [30] inserted two web proxies, one in the client's mobile device and one running within the wired network of the web server, to support both disconnected and asynchronous operations for mobile web browsers.

3.1.2 Caching Granularity

Caching granularity is concerned with the units of caching. Static web page caching uses URLs (Uniform Resource Locator) as the keys to cache web content in hashtables [27]. CASCADE [20] is an implementation of a distributed caching system for CORBA. It supports caching of "active objects", which include executable code and its dependant data. By wrapping each cached object into a proxy object, CASCADE enables the storage of CORBA objects in the clients' local memory or a place close to clients.

A semantic model for client-side caching in a client-server database system was introduced by Dar et al. [31]. Differing from the traditional tuple-and page-cache, the semantic cache maintains a semantic description of the cached data. Query processing then uses the semantic description to figure out which data are locally available, and which data should be fetched from the server. Figure 3-1 shows the four possible relationships between cache and query summarized by Dar et al. [31].

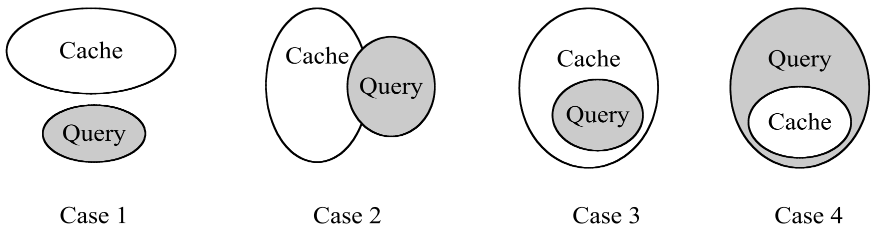


Figure 3-1. Possible relationships between cache and query [31]

Ren et al. [32] gave formal definitions of the semantic cache and related concepts. A semantic cache is modeled to include two parts: the index part and the content part. The content part consists of pages that store the tuples of semantic segments, and each page has a unique page id for identification. The index part keeps the following information:

- The semantic segment name S
- The set of participating base relations S_R
- The set of attributes S_A
- The predicate S_P
- The set of pages storing the semantic segment S_S
- The timestamp indication when the segment was last accessed S_{TS}

Table 3-1. An example of semantic index

S_{ID}	S_R	S_P	S_{A1}	S_{A2}
R1	Books	Type="Java programming"	Good	Second hand
R2	Restaurant	Type="Chinese"	5 miles	Excellent

Table 3-1 shows an example of the index part summarized from a sequence of service calls. Semantic caching has been applied for Location-Based Services (LBS) in mobile computing [33, 34], by setting up location-based semantic indices for the caching system. These semantic caching systems support semantic service discovery, such as "Give me all Chinese restaurants within 5 miles with good quality" (Table 3-1).

3.1.3 Consistency Policies

Cached items may become obsolete when the original data in the server are updated. Consistency strategies ensure the cached data are coherent with that in the server. For nomadic service clients, frequent network disconnection and movement across boundaries make consistency a challenging issue [25, 26].

Time-To-Live (TTL) is the expiration time of the cached item. Once the cached item has been retained past this expiration time, it will be invalidated. The invalidation based on TTL can only maintain weak cache consistency [27]. Wang [27] also pointed out two basic methods to maintain strong consistent web caching, namely the client poll method and the server push method. In the client poll method, the client requires a validation answer from the server before using every cached item, which leads to high cache-hit latency. In the server push method, the server monitors the states of cached items and sends invalidation messages to clients. Yin et al. [35] pointed out that the server push method is more efficient and costs less bandwidth than the client poll method for web sites containing large quantities of dynamically generated and frequently changing data. With the push method, the server can be a stateful server or a stateless server [20]. The stateful server keeps the information about its clients and their cache states. Every time a client caches an item, it sends a callback to the server. When data are changed, the server sends invalidation ids to clients based on the callbacks. The Coda File System [28] uses this method. This approach can minimize the client-server communication, but only works well within fast and reliable network environments. The stateless server has no information about its clients. It broadcasts updated data or invalidation ids to all clients. The clients' role can be passive - just wait for server's invalidation; or can be active - query the server for verification.

For mobile devices, states of both the server and the client need to be integrated after disconnections. The reintegration includes two steps: The first step is from the server to the client. The server sends a report containing invalidation information to the client, and then the client invalidates its cached items to keep consistence with the server. The second step is from the client to server. Upon reconnection, the client replays its queued requests to the server [26].

3.1.4 Replacement Algorithms

Another limitation of mobile devices is that they have limited memory, which results in a limited cache size. Since a replacement algorithm decides which cached items are to be kept in the cache and which ones are to be replaced, it affects the caching performance dramatically. LRU (Least Recently Used), LFU (Least Frequently Used) and SIZE are three traditional replacement algorithms, among which the most popular one is LRU [36]. Summarized by Wang [27] and Cao et al. [36] several replacement algorithms have been proposed for web page caching in order to minimize various cost metrics, e.g. average latency, byte hit ratio, and overall cost. For example, Log(size) +

LRU evicts the document having the largest log(size) and the least recently used; LRU-Threshold sets a certain threshold size for LRU; GreedyDual-Size (GD-Size) computes the cost of each object and evicts the one with the lowest cost.

3.2 Prefetching/Hoarding

Caching is a so-called lazy replication technology, by which an individual request is replicated (if not-yet-cached) for later reuse. If a response is labeled as cacheable, then a cache (e.g. client-side cache) is permitted to store it and reuse it for a later, equivalent request [27]. Prefetching/hoarding is an active replication technology, which preloads and hoards cacheable response items by predicting the future requests of clients [17] and, thus increases cache-hit ratio and reduces access latency. Prefetching has been applied to many areas, such as the World Wide Web (WWW), file systems and database systems. In the WWW, depending on the prediction algorithm and depth of fetching, prefetching reduces the client latency from 25% to 45% at the expense of increased bandwidth consumption and network traffic bursts [27]. Reference patterns observed by web servers, the client access history and the web log, can be effective sources of information for driving pre-fetching. The successful prefetching engine relies heavily on the prediction model and algorithm. Much research focuses on how to construct prediction models for prefetching.

3.2.1 Dependency Graphs

Dependency graphs were studied by Griffioen et al. [37, 38] to represent the probability of future file accesses based on past accesses. This technology is also used for web page prefetching by setting up the prediction model of web documents on the server [39]. The dependency graph is constructed to calculate the inter-relationship and conditional probability of document accesses, which are based on the statistical information about the client access pattern gathered by the system. Figure 3-2 shows an example of a web page dependency graph [39]. The nodes are the documents on the server that have previously been accessed. The arc between nodes, for example from node A to node B, represents that node B will be accessed within ω accesses after node A. The ω is the look-ahead window size. Each arc has a weight on it, which represents the probability of this access. In this example, the sum of the weights on arcs emanating from a particular node need not be 1.00 because the weight is not the probability node B will be accessed immediately after node A.

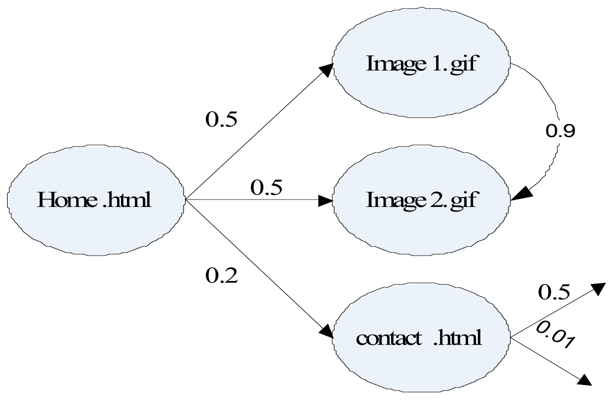


Figure 3-2. A dependency graph of WWW from [39]

Another similar prediction module [40, 41] for web page prefetching takes advantage of the web-browsing feature when constructing the dependency graph of web pages. Each web page has several links to other pages, and also clients may click the *forward* and *backward* buttons to access previous pages. Therefore, there is an arc from page A to page B if and only if there is a link to page B in page A. The weight is the access probability p(B|A), which is calculated based on the history and updated dynamically.

3.2.2 Workflow-based Technique

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." [42] A workflow can model business applications and/or behaviours of participants. The workflow management system enables defining, creating and managing the execution of workflow processes, in which developers define the order of tasks and the flow of data between applications.

Zhao et al. [43] addressed data management issues for large-scale, internet-based business applications, and discussed how to manage the business process as an information resource to improve the workflow efficiency. Figure 3-3 shows an example of an agent-document dependency graph.

As discussed by Zhao et al. [43], compared to web browsing, (1) workflow processes have an explicit model to define the sequence of tasks and the documents needed; (2) normally in workflow systems, the number of documents related to a task is small, so the data dependency tree is shallower than a web page which usually has more than 25 links; (3) the documents are prone to be changed because tasks can perform write operations. Using the workflow information, Zhao et al. [43] introduced an agent-document dependency graph to model the relationship and probability of each document on servers. Therefore, prefetching decisions can be made according to the model.

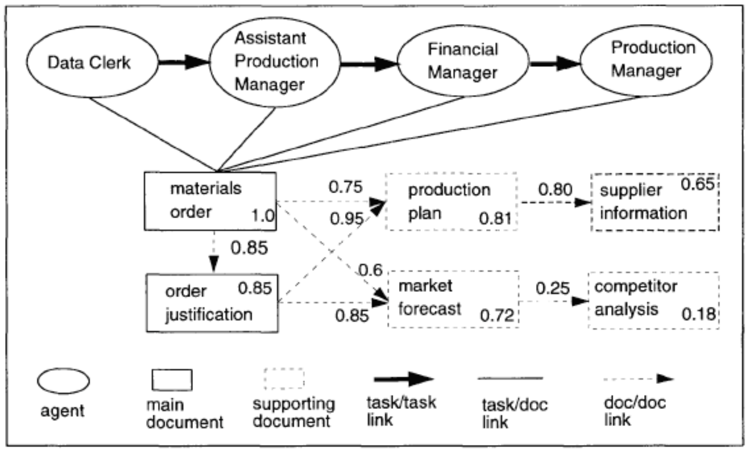


Figure 3-3. An example of agent-document dependency graph [43]

Jing et al. [44] proposed a background prefetching strategy based on the workflow specifics for mobile computing. The system predicts document access for next steps by analyzing the workflow specifics and prefetches them into the mobile device. Prefetching is performed in the background while the user is working on other documents. They also discuss how to make the prefetching strategy to be adaptive to the constrained bandwidth. The system decides the order of documents to prefetch based on user's reviewing speed, the size of document, and the available bandwidth. For example, the system prefetches the shortest document for the user to review when the bandwidth is low. Otherwise, if bandwidth is high and user's processing speed is getting slow, the system prefetches a larger document.

3.2.3 Hint-based Prefetching

The prefetching/hoarding model and algorithm can be based on application level and user profile level hints about the future access patterns. Transparent Informed Prefetching was studied for the Coda file system by Patterson et al. [45, 46]. TIP enables Coda to prefetch needed data and optimize resource utilization by analyzing hints in the application code given by programmers. For example, a programmer adds a hint as "I will read file F sequentially with stride S" in the code, and then TIP prefetched file F to reduce file access latency. Hints can also be found in the user profile. Sharifi et al. [47] proposed a distributed application prototype for enabling homecare workers to communicate and access information with mobile devices. This prototype could predict the information needs of nomadic users by tracking users' current context and setting a user model in this specific domain.

3.2.4 Prefetching Issues

When to prefetch: if items are prefetched too early and the cache capacity is limited, the prefetched items may replace cached items that will be accessed before them. Moreover, if the items are not static, they may change after being prefetched. This can result in an extra load to invalidate these items or providing obsolete data to the client. If the items are prefetched too late, they may not be able to arrive at the cache before the client accesses them.

Prefetching threshold: If the caching system knows exactly the next items the client needs, it can always preload these items in advance and the clients will not need to worry about latency and the extra cost of prefetching. However, predicting the future behaviors and requirement of clients is not entirely reliable and some prefetched items may never be used. The side effects of this unreliable predictability could be extra bandwidth consumption and server load (e.g. in WWW [41]). The prefetch scheme proposed by Jiang et al. [41] has not only a prediction module, but also a threshold module. The threshold module calculates the prefetching threshold dynamically based on bandwidth consumption and network latency to optimize the tradeoff between the extra cost of prefetching and the latency cost of non-prefetching.

3.3 General Review on WS Caching

WS is "a software system designed to support interoperable machine-to-machine interaction over a network" [4]. With the introduction of the three fundamental components – SOAP, WSDL, and UDDI, the WS framework has emerged as a promising realization of SOA. Implementing an SOA with WS takes advantage of mature Web technologies and the basic internet infrastructure, which make the implementation pervasive, simple and platform-independent. As caching for WS is a new research area, so far there has not been much research work on it. Table 3-2 shows previous research on WS caching.

Table 3-2. Evaluating existing caching system for WS

Systems	Location	Purpose	Metadata	Operations	Cached Item	Consistency	Replacement Algorithm	Prefetch
Takase et al.[48]	Client	Static	From name	Read	SOAP, parsed SOAP	TTL	N/A	N/A
Devaram et al[49]	Client	Static	N/A	Read	SOAP template	N/A	N/A	N/A
Terry et al.[25]	Client	Mobile	WSDL annotation	Read, Write	SOAP	N/A	N/A	N/A
CRISP [26]	Client	Mobile	User GUI	Read, Write	SOAP	Dependency matrix	LRU	N/A
Friedman [22]	Proxy	Ad-hoc network	N/A	N/A	Data and code	Combination	N/A	N/A
WebSphere [29]	Server	Static	From name	Read	SOAP	TTL	N/A	N/A
Seltzsam [50]	Client	Static	WSDL annotation	Read	Semantic SOAP	TTL	N/A	N/A

Because considerable processing time (up to 40% in some cases [49]) is spent on XML/SOAP message de/encoding, the SOAP protocol is a bottleneck of WS. Devaram et al. [49] proposed to cache SOAP request message templates in the client side and generate SOAP requests by using the templates, and thus reduce the de/encoding XML/SOAP cost. Takase et al. [48] described a caching system implemented as a WS client middleware that can be deployed transparently to the existing WS framework. It also evaluated the impact of three optimization methods - caching XML messages, caching SAX event sequence and caching read-only objects - on the performance of this caching system.

Terry et al [25, 51] discussed the issues of caching WS for mobile devices. CRISP [26] was a client-side SOAP caching system for nomadic WS clients, which aimed to improve the availability of services during disconnection. They recognized that the WSDL document does not contain sufficient service description for supporting WS caching and additional metadata is needed. Friedman [22] applied CORBA caching techniques to WS caching in ad-hoc networks, and discussed that caching services must be able to cache both data and code. This approach may be feasible for tightly-coupled inter-organizational interactions. However, for the loosely-coupled services in SOA/WS, the applicability of this approach can be doubted.

IBM WebSphere [29] supports caching SOAP response messages on the server side. When SOAP response messages are generated from the cache, execution time spent on de/encoding the SOAP messages is saved. This also reduces server workload stress by avoiding the expense on computation of the same frequent requests.

Seltzsam et al. [50] discussed how to apply the semantic caching schemes to WS caching on the SOAP protocol level and focused on read-mostly services, such as query-like interfaces to access product categories. This work added the caching-relevant semantics as annotations to WSDL specifications. These annotations were used for mapping SOAP requests to predicates, fragmenting responses, and reassembling responses. One obvious advantage of the semantic cache is that it needs much less memory than SOAP caching. Moreover, in the heterogeneous environment of WS, the semantic cache is flexible when considering physical storage structure and data format. The disadvantage of semantic cache is that it needs extra processing to convert semantic cache items to or from SOAP messages.

3.4 Metadata for Caching

In order to cache messages exchanged between services, the caching system should be able to identify the cache-ability of the services. Services can be essentially categorized into two types: state-reading (`read`) and state-altering (`write`). The response message from a state-reading service is cacheable as the invocation of the service is just for querying information, while the response message of a state-altering service is considered uncacheable because the purpose of invoking this service is to change some states of the server. Then comes the issue of invalidation. When a `write` service is invoked, it may change the states related to the previous `read` services cached. Therefore, in order to maintain the consistency between the caching system and the server, the caching system should be able to identify the dependency of each `read`/`write` Service [25, 26].

WSDL only specifies syntax, but not semantics, of services. It describes service functionality in terms of inputs, outputs and operations to enable clients to consume services without knowing the internal implementation and data operations. However, the information from the WSDL specifications is limited when considered as a resource of the semantics for caching services [25, 26]. In order to give the cache manager a reasonable understanding of services, Terry et al. discussed the need for service metadata, which contains information about cache-ability, lifetime, play-back, and default-response.

CRISP [26] provided clients a GUI to input metadata, based on which it generated a XML specification for the caching system.

3.4.1 Web Service Standards

In order to use WS to address more difficult problems and build more complicated applications, the WS community has done significant work on introducing various WS-related specifications and standards to provide secure, reliable, and transacted interoperability. Figure 3-4 shows the stack of current and emerging WS standards.

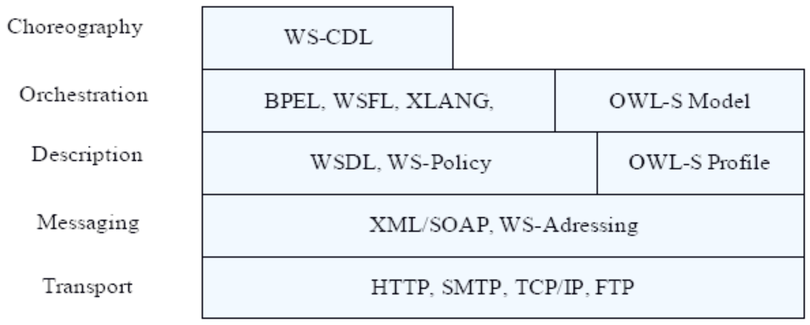


Figure 3-4. WS standards stack: key elements

3.4.2 WS Orchestration Language

One concern of SOA is Business Process Management (BPM). A service is composed of "activities" that can be integrated into applications and business processes. SOA facilitates the composition and management of large-scale enterprise and Business-to-Business (B2B) application systems by introducing BPM methodologies and technologies. Business processes consist of a set of human and machine-based activities, which are performed according to business logic to realize a business objective or policy goal. Figure 3-4 shows several existing orchestration languages, among which BPEL is standardized as the orchestration language of WS.

BPEL is an XML-based WS orchestration language heavily relying on WSDL. It defines variables using WSDL message types and XML-Schema data types, uses WSDL <portType> to reference external services to define interactions between business partners, and in turn, the business process composed by BPEL can be exposed as a new Web service using WSDL. BPEL "defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners." [14]

BPEL is designed to enable *programming in the large*, which describes the high level logic and state of interactions among services, such as when to send a message, and which service to be invoked based on a condition.

Table 3-3. BPEL structured activities [14]

Activity	Description
Sequence	A set of activities performed sequentially
Flow	A set of activities performed in parallel
While	Iterate execution of activities until the given Boolean condition is violated
Pick	A set of events is being executed. Selects the activity corresponding to the first event finished.
Switch	Support conditional behaviors as case, otherwise branches
Link	Defines a control dependency between a source activity and a target

In a BPEL specification, the elementary components are either atomic or composite WS. BPEL defines both basic and structured activities. <invoke>, <receive>, and <reply> are basic activities for defining communications with external WS: <invoke> is to invoke an operation defined in WSDL, <receive> is for waiting for a response from other services, and <reply> is to send a response to other services. Also there are several basic activities for local behavior: <assign> is to copy message data, <throw> is for throwing an exception in the execution, <terminate> is to terminate the entire process, and <wait> is to specify a delay [14]. Structured activities (Table 3-3) are for controlling the execution flow. The meaning of these activities is similar to those in the traditional programming languages.

3.4.3 Web Ontology

The Semantic Web community has produced a variety of ontology languages for providing semantic service descriptions to enable machine understandable information exchange. The goal of the Semantic Web and WS is to enable service consumers to locate, interoperate, select, deploy, and monitor WS automatically. "The challenge of the Semantic Web is to find a representation language powerful enough to support automated reasoning but simple enough to be usable". Among the best known are DARPA Agent Markup Language + Ontology Inference Layer (DAML+OIL [52]), Ontology Web Language (OWL [53]), and OWL-S. These ontology languages are designed to adapt to the frequently changing web content. Figure 3-5 gives markup language and web ontology stack of W3C recommendation [54].

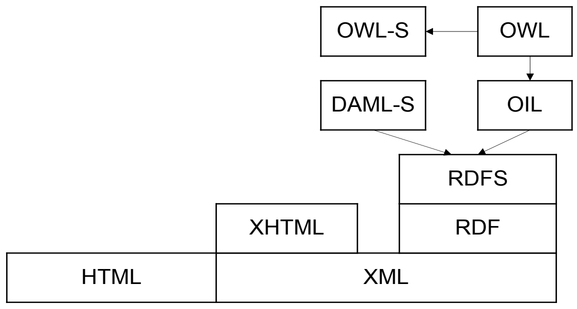


Figure 3-5. Markup languages and web ontology

"The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web." [55] It is based on the idea of expressing simple statements about web resources. An RDF model offers an XML-based syntax (RDF/XML) to make machine process-able statements. Figure 3-6 shows a simple RDF statement that contains an assertion (Subject-Predicate-Object statement). RDFS (RDF-Schema) [56] is a semantic extension of RDF, and it provides a vocabulary for describing properties and classes of RDF resources.

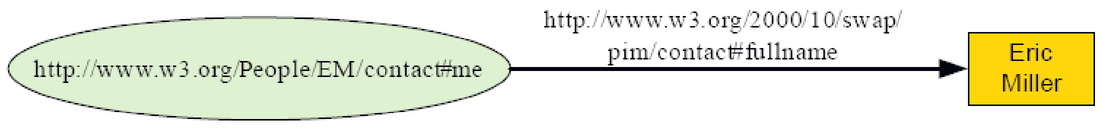

http://www.w3.org/People/EM/contact#me
http://www.w3.org/2000/10/swap/pim/contact#fullname
Eric Miller

Figure 3-6. A simple RDF statement [55]

"An ontology is a formal, explicit specification of a shared conceptualization". OWL is web ontology built on the top of RDF/RDFS and is derived from DAML+OIL. OWL and OWL-S are recommended by W3C for publishing and sharing ontology of Semantic Web and WS. It has three important elements [54]:

Description Logics (DLs): OWL-DL has formal foundations and enables reasoning. Reasoning allows the machine to derive new data from the existing data, check the consistence and find semantic similarity in spite of syntactic differences.

Frame-base system: OWL incorporates essential modeling primitives for modeling aspects of a domain. Classes in OWL are defined as subclasses of property restriction. This makes OWL flexible for describing the frequently changing web content.

Web Standards: OWL uses XML and RDF-based syntax for information representation, which makes OWL understandable and exchangeable by machines.

OWL-S is defined by OWL, which provides a set of ontologies to describe WS. OWL-S documents provide essentially three parts of service description: the service profile – what the service does, the service model – what happens when the service is carried out, and the service grounding – how a service may be accessed [16]. The service model gives the detailed information about input, output, precondition, and effects of each atomic service. Thus, the service consumer can use the service model to control the interaction with the service.

3.5 Summary

Previous work in caching and prefetching, along with work in WS, provide a starting point for research on SOA/WS caching. There are several issues, such as cache location, replacement algorithm, cache consistency and semantic cache that need to be considered in SOA/WS caching.

Metadata is a crucial issue for caching services in SOA/WS, because the semantics, of services is not provided by the WDSL definition. How to model services in a comprehensive and efficient way is still an open question. The proposed service metadata only specifies attributes of each individual service and may not be comprehensive enough to describe inter-relationships among services. Moreover, no study has been done on how to use metadata or how the granularity of metadata impacts the caching system performance. One aim of this research is to identify whether the WS standards and ontology (e.g. RDF/RDFS, OWL-S) can provide a standard way to specify service metadata.

Although prefetching has proven to be an effective way to improve the performance of caching systems, this technique has not been applied to the SOA/WS caching area. The open question is how to model client behaviors in order to anticipate future service requests and how to apply adaptive prefetching policy to deal with the changing environments. WS orchestration languages (e.g. BPEL) offer a way to integrate services into workflows. Combining the workflow-based prefetching technique with the use of the cache, this research will identify the possibility of applying the prefetching technique to SOA/WS caching based on BPEL specifications.

CHAPTER 4

MODEL-DRIVEN DUAL CACHING

Chapter 2 discussed the problems faced by the nomadic clients in SOA and the necessity to use a dual-caching system to overcome the problems. The focus of this research is on how to use models (service, client and wireless network) to enable caching and prefetching strategies in the Model-Driven Dual Caching (MDDC) system. The general architecture of MDDC is presented in Section 4.1. Section 4.2, 4.3, and 4.4 describe service metadata, the client workflow model, and the wireless network model, respectively. Finally, Section 4.5 details the functionalities offered by MDDC.

4.1 Model-Driven Dual Caching (MDDC) System

Figure 4-1 shows the architecture of the proposed MDDC system. Two transparent caches, one on the client side and one on the provider side (Section 2.3), are required to overcome the loss of connectivity during the sending or receiving of the SOAP traffic.

The transparent client-side cache (CSC) shields the service consumer from the sudden loss of connectivity during a service call. To the client, the CSC appears as a local proxy residing on the mobile device. Whenever the client issues a request, the CSC first tries to serve the request from its cache. If this fails, the request is sent to the provider-side cache (PSC). The response will be cached on the CSC (if metadata indicates that it is cacheable).

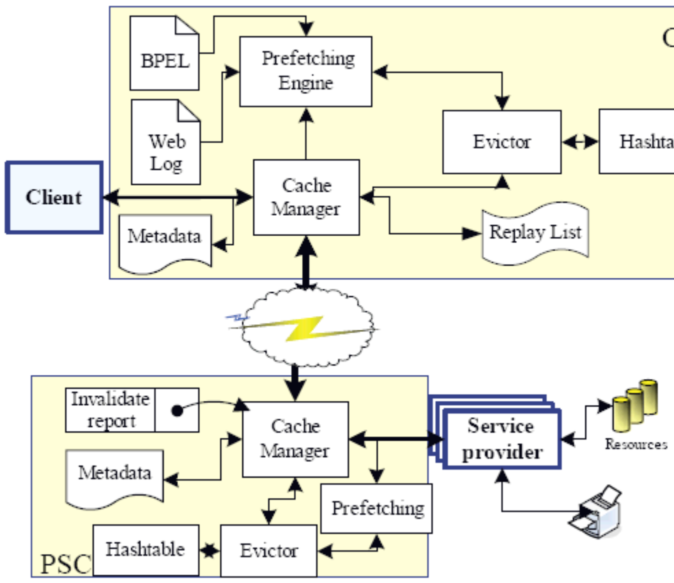


Figure 4-1. Dual-Caching system architecture

The PSC resides on a proxy server which has a reliable connection to the service provider. The incoming requests from the CSC are sent directly to the provider. Responses from the provider are first cached in the PSC and then sent to the CSC. In the case that the CSC cannot be reached, the PSC will wait for the CSC to issue a reconnect message upon which, the PSC will resubmit all queued responses.

The CSC & PSC use a cache-manager as the main coordination component. Both use a hashtable to store the request-response pair that uses the request as the key and the response as the data. The hashtable is governed by an evictor component that is responsible for the management of the hashtable, which includes replacement algorithms and a consistency policy. When the hashtable has reached its maximum capacity, the evictor will evict items based on a replacement algorithm (e.g. LFU). The evictor is also responsible for invalidating cached items based on TTL and invalidation reports from the PSC. The prefetching component is responsible for prefetching service response messages based on a prefetching algorithm. The prefetching engine takes advantage of idle time and uses background threads to send prefetching requests to the service provider. On the client side, the prediction algorithm is based on utilizing BPEL files that describe the client workflows. On the server-side, the prediction algorithm is based on the prediction dependency relationship among services. The PSC analyzes the known relationships between the services and creates invalidation reports that are broadcasted to clients to invalidate stale data in the CSC. The invalidation report is based on the invalidation dependency relationship. When the server receives a `write` operation, it checks the invalidation dependency to decide which `read` operations are to be invalidated. The invalidation report containing all invalidation ids is piggybacked as a part of the SOAP header to clients in a fixed period.

4.2 Service Metadata

The service model is internally viewed as service metadata. Chapter 3 discusses issues on caching and prefetching, e.g. cache-ability, cache replacement algorithms, and cache consistency. In order to support caching and prefetching request/response messages, two fundamental questions must be answered:

- Which service requests are cacheable?
- Which operations do change states of the server? And what influence do they have?

However, the services description specified by WSDL does not have information to answer these questions. Therefore, a service model – service metadata, is proposed.

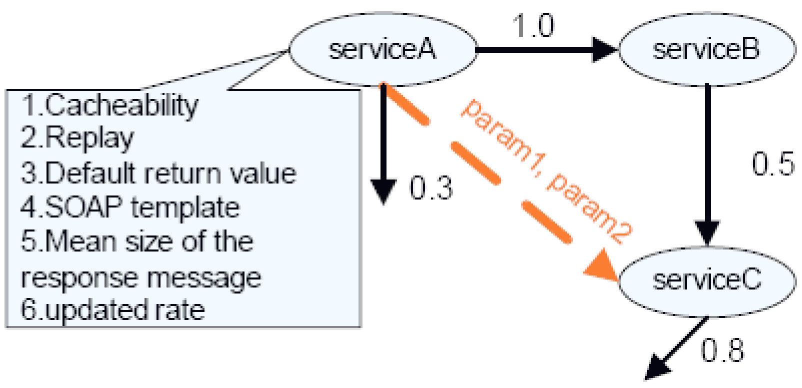


Figure 4-2. Service metadata graph

Figure 4-2 depicts the service metadata graph that includes two levels. The core-level metadata describes properties of each individual service:

- Cacheability: services are classified into two broad categories: cacheable services and uncacheable services. If the service is a state-reading (`read`) operation, that is, invoking this service does not change any states of the server, this service is identified as cacheable. Otherwise, if the service is a state-altering (`write`) operation, it is identified as uncacheable.
- Replay: if the service is a state-altering (`write`) operation and its behavior could not be discarded upon reconnection, it is tagged as replay. Replay service requests can be queued in a replay list in the CSC for later processing.
- Request SOAP message template: The prefetching engine generates SOAP request messages by inserting the parameters to the SOAP template. Generating SOAP requests from the templates reduces the overhead caused by SOAP message processing.
- Default return-value: this value is to give the client a meaningful return value when a disconnection occurs.

- TTL (Time-To-Live): TTL is the expiration time of the cached item. Each cached item is associated with a TTL value upon creating, and invalidated after this expiration time. The invalidation based on TTL can only maintain weak cache consistency.
- Mean size of the response message, fetching cost and updated rate: First, the size of messages exchanged among services can vary significantly due to which service is requested and what response information they carry. Second, services are functional interfaces related to background databases and server resources. Different services and input messages create different network latency and server workload stress. Third, compared to web pages, services are more dynamic. Data can be updated from the server and clients at different frequency. This leads to different update rates and invalidation cycles of services. These properties are for modeling costs of invoking the service, which provide the necessary information for applying the adaptive prefetching policy and cost-based replacement algorithms, e.g. GD-Size, to the caching system.

The detailed-level metadata is for describing relationships among services. Two kinds of dependency relationship are defined:

- Prediction dependency: The solid black arc (Figure 3-2) between two services indicates that the second service will be accessed after the invoking of the first one within a threshold time. This dependency is inspired by file dependency [37, 38] and web page dependency [39]. The value of the arc represents the probability of the future access. This relationship indicates the prediction dependency observed by the server. Based on the prediction dependency, server push technology can be applied on the server or a proxy server. Note that the prediction dependency relationship is between `read` services, as only `read` services can be prefetched and cached.

- Invalidation dependency: After a state-altering (`write`) service is invoked, it may change some states of the server. As a result some cached request/response messages relating to these states become stale, and thus must be invalidated. The dotted arc (Figure 3-2) pointing from a state-altering (`write`) service operation to a state-reading (`read`) service operation represents this invalidation dependency. The parameters on the arc represent the conditions of this dependency, because what states the service will change also depends on the input parameters being passed [51]. The invalidation dependency generates invalidation ids in two levels of granularity: the coarse-grained invalidation ids are generated only based on the service operation name, while the fine-grained ones include dependency parameters. The invalidation dependency is for keeping strong consistency of the caching system.

RDF/RDF-Schema is chosen for describing service metadata. The RDF language is a W3C standard for representing metadata of web-based resources. Typically, RDF uses an object-attribute-value triple (O→A→V) to represent that "object O has an attribute A with value V". Similar to the XML-Schema of XML, RDF-Schema offers developers a vocabulary-definition facility to specify a particular vocabulary (e.g. declaring property names and class names) [57]. Describing service metadata with RDF/RDFS has the following advantages:

- Universal expressive power, syntactic interoperability and semantic interoperability. [57]

- Reusability and flexibility for the frequently changing Web content.
- OWL/OWL-S is built on the top of RDF/RDF-Schema. It is easy to map a RDF/RDFS specification to an OWL/OWL-S specification.

Service metadata is provided by the service provider that has a comprehensive understanding of its functionality. The caching system reads and maintains the metadata in local memory. The costs of the service in metadata, i.e. the mean size of the response message and the updated rate, are dynamically updated upon the receiving of new requests by the caching system. Appendix A provides an example of service metadata defined with RDF/RDFS.

4.3 Client Workflow Model

Several WS orchestration languages have been introduced by the WS community and WS vendors, e.g. BPEL and OWL-S (shown in Figure 3-4). The motivation of these languages is to integrate loosely coupled services into a business process workflow. A business process workflow contains information about activities the client will take, the sequences of the activities and the relationships among them [58]. Hence, future behaviors of the client can be predicted by analyzing the workflow specification.

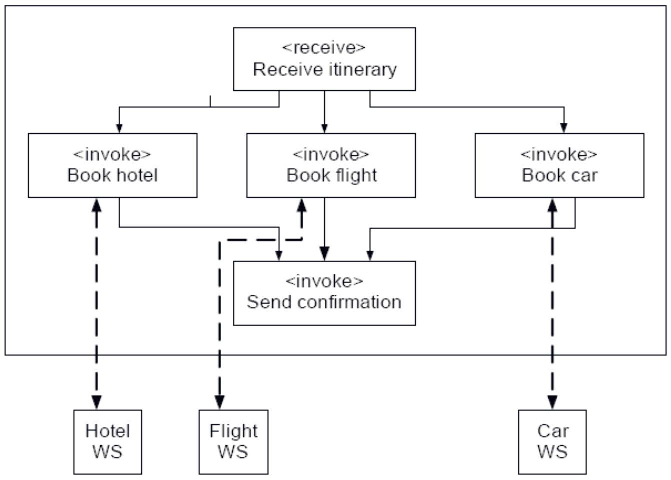


Figure 4-3. An example of BPEL process [58]

Figure 4-3 depicts an example of a BPEL process workflow, which is a travel-booking process. It starts by receiving a customer's itinerary, then attempts to invoke hotel, flight, and car booking services, and ends by sending a confirmation message to the customer [58].

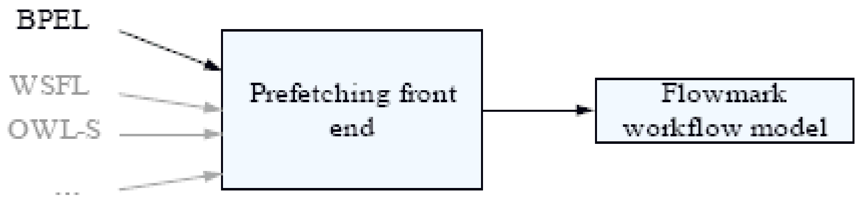


Figure 4-4. Model transforming

Since each language has its own way to specify the logic and data streaming, a uniform workflow model is needed (Figure 4-4). A FlowMark workflow [59] is chosen for modeling client behaviors. The advantage of this approach is that it keeps the model consistent and simple.

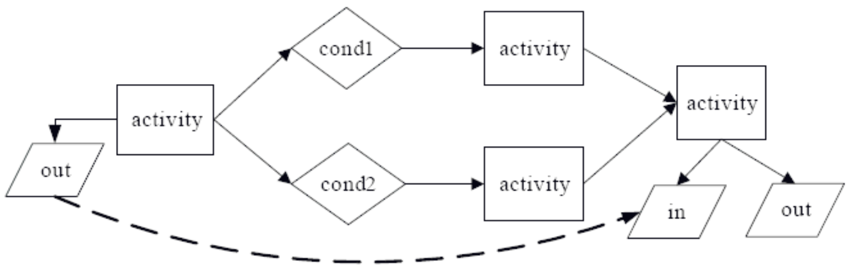


Figure 4-5. Workflow Representation of Business Process

From Figure 4-5 we can see that the FlowMark workflow includes five components: processes, activities, control connectors, data connectors, and conditions [59]:

- Process: The whole graph specifies a process. It describes the sequence of steps involved to accomplish a given goal.
- Activity: Each square node in the process graph represents an activity to be executed.
- Control connector: The solid arcs are control connectors, which specify the flow of control.
- Data connector: Activities can have data containers, which represent the input/output messages. The dotted arcs between data containers are data connectors, which specify the flow of information from one activity to another.
- Condition: The diamonds on control connectors are conditions, which are used to specify certain conditions to determine the work flow direction.

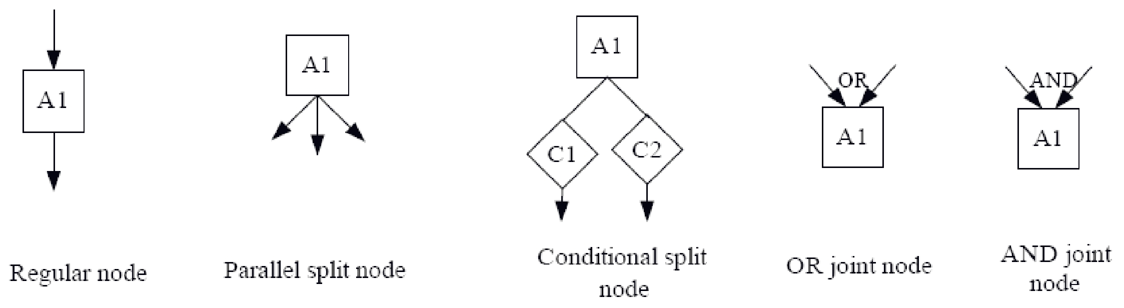


Figure 4-6. Nodes in the FlowMark workflow model

When analyzing the flow of control for prefetching, five different nodes should be identified. Figure 4-6 shows the graph of each node.

- Regular nodes: these are the simplest nodes in a workflow, since they have only one incoming and one outgoing control connector. The workflow can continue smoothly through regular nodes to the next step.
- Parallel split nodes: these have more than one outgoing control connectors. These nodes start multiple branches of workflows that can be executed in parallel.

- Conditional split nodes: they have more than one outgoing control connectors and each has a condition on it, which branch is chosen depends on the condition. One issue for these nodes is how to calculate the probability on which branch is chosen. The simplest way is to assign all branches an equal probability (i.e. 1/number of branches). Hence, prefetching algorithms and policies are needed for this point.
- *OR* joint nodes: these have more than one incoming control connectors, which has an *OR* operator. The *OR* operator means the incoming control connectors do not need to be synchronized. Whenever a branch of activities is finished, the activity can be executed without waiting for other branches.
- *AND* join nodes: these have more than one incoming control connectors, which meet an *AND* operator. The *AND* operator means the incoming control connectors must be synchronized. Therefore, the activity should be blocked until every incoming branch of activities is completed.

Another important issue is the flow of data. A service is a function that takes request messages as input parameters, performs actions, and then converts output results to a response message sent to other services [59]. As shown in Figure 4-5, an activity's output can be the input of another activity. An activity can not be executed if its input parameters are not available.

Given a BPEL process specification and the related WSDL files, a FlowMark workflow model can be constructed. The basic activities of BPEL, such as <invoke>, <receive>, <reply>, <wait>, <empty> and <terminate>, can be translated into activity nodes. BPEL provides structured activities that support complex workflow patterns (WPs)

[60]. Appendix B shows how to translate BPEL structured activities into a FlowMark model.

The input/output parameters of services can be translated to in/out data containers attached to activities. In BPEL, <assign> supports data copy between variables, and also provides the ability to "construct and insert new data using expressions" [59]. The <assign> activity can be translated to a data connector between out/in data containers of activities in a FlowMark workflow model.

Combining workflow and dataflow, the model can be further optimized. For example, if there is no dataflow relationship between each <sequence> activities, they can be converted to parallel activities.

4.4 Wireless Network Model

As discussed in Section 2.2, one constraint of mobile devices is that they have to rely on the wireless network for communication. Voluntary or involuntary disconnections and fluctuating bandwidth are not considered as exceptions but rather as the norm of using wireless communication [12]. Moreover, when a nomadic client is moving around, it may switch from one wireless connection to another. Presently the wireless network connections available on the market have different cost and capacity. The purpose of the wireless network model is to provide connectivity information based on a client's location and schedule. The wireless network model allows applications to discover the current wireless environment and to predict changes in the near future, and thus the application changes its policies to be adaptive to the changing environment by taking advantage of the information. To cope with the wireless network changing consecutively, the wireless network model represents the wireless connectivity in terms of time period. For each period, the following properties are recorded:

- Start time: indicates the start time of this wireless network connectivity.
- Finish time: indicates the finish time of this wireless network connectivity.
- Connectivity: indicates whether the mobile device is connected or disconnected.
- Bandwidth availability: indicates the available network bandwidth of this wireless network connectivity.
- Bandwidth cost: indicates the cost of transmitting 1 kilobyte (KB) of data.

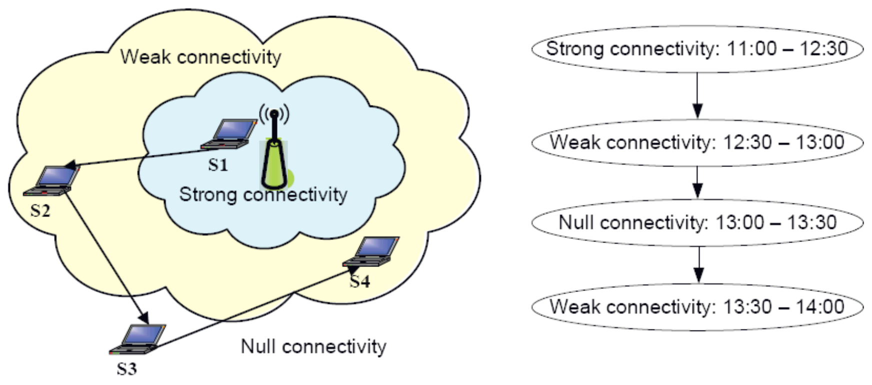

Weak connectivity
S1
S2
Strong connectivity
S4
S3
Null connectivity
Strong connectivity: 11:00 – 12:30
Weak connectivity: 12:30 – 13:00
Null connectivity: 13:00 – 13:30
Weak connectivity: 13:30 – 14:00

Figure 4-7. An example of the wireless network model

Figure 4-7 gives an example of the wireless network model. The right side picture shows a scenario that a nomadic client moves around from time to time. For example, in location S1, from 11:00 to 12:30, the nomadic client has strong connectivity as it is close to a base station. After it moves to location S2 from 12:30 to 13:00, it has weak connectivity. The picture on the left side shows the wireless network model of this scenario.

An ideal situation for generating the model is that the mobile device has a map of the current environment, and then the wireless connectivity of different areas can be calculated based on the distance to the base stations and obstacles around. The client's current location and next location can be provided by the client or found in its schedule.

Another way to get the map and the client's location is to attach a Global Positioning System (GPS) receiver to the mobile device. Assuming the laptop is equipped with a GPS application that can provide a detailed map. If the GPS map is able to show a tunnel ahead on the road, the caching system can calculate when the car will enter that tunnel and service requests will be prefetched as a result.

4.5 Prefetching/Hoarding Strategy

As discussed in Section 3.2, caching performance (e.g. cache-hit ratio and latency) can be improved by prefetching/hoarding technique. Furthermore, prefetching/hoarding of needed service messages ahead of disconnections gives the client an illusion of a constant connection. By combining the three models (service, client and network), MDDC is able to predict the environment changing and prefetching/hoarding the client's next cacheable activities. However, if the client model is not 100% accurate, prefetching/hoarding can cause network overheads and server load increase. In order to achieve better performance and benefit/cost ratio, the Dual-Caching system is configured to have different strategies under different wireless connectivity. Table 4-1 shows the related parameters of prefetching.

Table 4-1. Related parameters of prefetching a service

Parameter	Unit	Comment
Prediction weight (P)	Unit	Probability of being invoked in the near future
Available Bandwidth (B_a)	KB	Bandwidth available
Bandwidth cost (B_c)	Dollar	Bandwidth cost per KB
Mean message size (S_m)	KB	Mean message size in KB
Updated rate R_u	Unit	Updated rate by the server and `write` operations
Latency (L_a)	Million Second(ms)	The service will be prefetched before this time length
Bandwidth waste	Dollar	Bandwidth waste limitation per request set by

limitation (B_w)		the client

4.5.1 Strong connectivity

When the client has strong connectivity, it appears no different to the service provider than clients using wired devices. The main concern at this situation is how to achieve a better performance but in a reasonable cost. Based on the parameters in Table 4-1, bandwidth waste (W_B) of prefetching a service is:

$$W_B = (1 - P + R_u) \times S_m \times B_c$$

Where, P is the probability of the service in the near future, R_u is the update rate for the service (rate of `write` operations). Therefore, $(1 - P + R_u)$ calculates the probability that the prefetched service will not be used or, will be changed by others and has to be prefetched again. S_m is the mean message size, and B_c is the bandwidth cost per KB. The whole formula calculates the cost of the waste bandwidth.

If the bandwidth waste limitation (B_w) is set by the client, then the prefetching weight threshold (P_t) can be determined using the above formula:

$$P_t = 1 - \frac{B_w}{S_m \times B_c} + R_u$$

Therefore, the prefetching engine only prefetches services whose prediction weight (P) is larger than the prefetching weight threshold (P_t).

4.5.2 Weak connectivity

Weak connectivity is characterized by intermittent connection and low bandwidth, which result in high network latency and message losses. The main concern in this situation is how to reduce the network latency and support seamless access. The latency of a service request can be estimated as:

$$L_a = S_m/B_a$$

Therefore, if the service can be prefetched ahead of time L_a, the bandwidth limitation is shielded by MDDC. When the wireless network model detects a disconnection, MDDC prefetches services before the disconnection. How many services are to be prefetched depends on the length of the disconnection. Besides prefetching/hoarding, MDDC is configured to have the following functionality to shield the limitations of the wireless network:

- Because the wireless connection can be interrupted at any stage of the message's transmission (e.g. four interruption scenarios in Section 2.2), the CSC converts synchronous requests to asynchronous requests and put them into the replay list.
- Under weak connectivity, the client may lose the invalidation report from the PSC. The client requests invalidation reports from the service provider upon reconnection.
- The prefetching strategy should be able to adapt to the constrained bandwidth. The system decides the order of prefetching items based on the client's reviewing speed, the mean message size of services, and the available bandwidth. For example, the system prefetches the shortest document for the client to review when the network bandwidth is low. Otherwise, if the bandwidth is high and the client's processing speed is slowing, the system prefetches a larger document.

4.5.3 Null connectivity

Null connectivity can be caused by voluntary disconnections (e.g. to turn off the Wi-Fi to save battery) and involuntary disconnections (e.g. no signal coverage). The main concern in this situation is to enable the client to continue its work while in a disconnected state. If the wireless network monitor predicts null connectivity is about to occur, the CSC prefetches as much as possible according to the client workflow model. During null connectivity, when a cache-miss happens, the CSC creates a default-value using the *default-return value* in service metadata, and thus the client can continue its logic. `Write` operations and cache-missed `read` operations are put into the replay list. Upon reconnection, the CSC discards all cached items and sends operations in the replay list to the service provider.

The CSC provides a GUI for the client to give instructions and organize the caching system. The existing work on WS caching emphasizes the transparency of the caching system. The caching system should let clients give instructions, such as when to disconnect, what the client wants to prefetch, and to cancel operations in the replay list during disconnection. This information will be helpful to improve the performance of the caching system. For example, in the travel-booking scenario, before the flight takes off, the business person has to turn off his laptop. He can instruct the system when he will turn off the network connector and what information he wants to download, and thus he can continue his work on the flight.

CHAPTER 5

EVALUATION WITH SYNTHETIC WORKLOADS

This chapter presents the experiments and results for evaluating the Model-Driven Dual-Caching (MDDC) framework with synthetic workloads. The first section outlines the experimental setup. The second section describes the underlying concepts for generating the workloads. The evaluation results then presented are divided into three sections:

- The experiments in Section 5.3 investigate the caching performance under different caching strategies and levels of wireless connectivity.
- Section 5.4 presents the results for evaluating the prefetching overheads and benefits in different conditions.
- Section 5.5 presents the effect of models with different accuracy on the performance of the MDDC framework. The models include the client workflow model, wireless connectivity model, and service metadata.
- Finally, Section 5.6 is a summary of all the evaluation results.

Performance comparisons are made based on the values of the following parameters:

- Cache-hit ratio: cache-hit ratio is a standard performance measure for evaluating caching systems. A higher cache-hit ratio means more responses are served from the cache without being sent to the server. On a cache hit, the bandwidth consumption and server loads are reduced by re-using data in the local storage. This parameter includes both the CSC & PSC cache-hit ratio.

- Latency: latency is the client-perceived time delay between the moment the client sends a request and the moment it receives the response. Latency in the Dual-Caching system is broken down into four components: the time spent in the CSC, the network delay, the time spent in the PSC, and the server-response time. Latency may include these four components or a portion of them, depending on where the response is originated.
- Throughput: throughput is expressed in interactions per second, which is equivalent to the number of interactions in the workload divided by the workload duration. This parameter measures the improvement in processing capacity and the speed of the MDDC framework.
- Bandwidth consumption: bandwidth consumption measures the bytes transmitted through the wireless network. In a wireless environment, the bandwidth cost is high compared to a wired LAN. This parameter is used to determine the overheads and benefits of different caching and prefetching strategies.

5.1 Synthetic Workloads

Because no WS traces are available, the workloads used in the experiments are synthetic. In order to simulate real world client behaviors, three workloads are generated based on the following specification:

- The percentage of `read/write` operations in the workloads is according to the TPC-W E-commerce benchmark: shopping, browsing, and ordering. (Table 5-1 shows the percentage of `read/write` operations in each workload).

- Service request arrivals follow the Poisson distribution (see Section 5.1.2).
- The web resource popularity follows a Zipf-like distribution (see Section 5.1.3).

5.1.1 TPC-W Benchmark

The TPC-W (Transaction Processing Performance Council - Web) E-commerce benchmark [61] has been widely used for measuring the performance of E-commerce systems. TPC-W defines two kinds of web interaction: browse and order. The browse behavior involves browsing the web site and querying the database, e.g. searching for new products, best sellers, and product details. The order behavior updates the database, e.g. loading shopping carts, and registering customers. By varying the percentage of the browsing and ordering behaviors, TPC-W defines three different kinds of workload: shopping, browsing, and ordering. Table 5-1 describes the distribution of client behaviors in the workloads.

Table 5-1. Read/Write percentage in each workload

Workload Type	Read	Write
Browsing	95%	5%
Shopping	80%	20%
Ordering	50%	50%

5.1.2 Poisson Distribution

By studying current network traces, Karagiannis et al. [62] observed that network traffic can be well represented by the Poisson model. The typical Web request distribution follows the Poisson distribution.

The Poisson distribution [63] represents the number of accesses per unit time, whose inter-arrival time follows an exponential distribution. Its probability function is:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

In this function, λ denotes the mean of the distribution. Figure 5-1 shows the Poisson distribution graph. “The horizontal axis is the index *k*. The function is only non-zero at integer values of *k*.” [63]

Figure 5-1. Poisson distribution [63]

5.1.3 Zipf’s Distribution

The Zipf's distribution is generally used to refer to frequency distributions of "rank data". For I = 1, 2… R, Let $P_R(i)$ denotes the popularity of the ith-ranked item, the $P_R(i) = \Omega/i^{\alpha}$, where

$$\Omega = \sum_{i=1}^{R} 1/i^{\alpha}$$

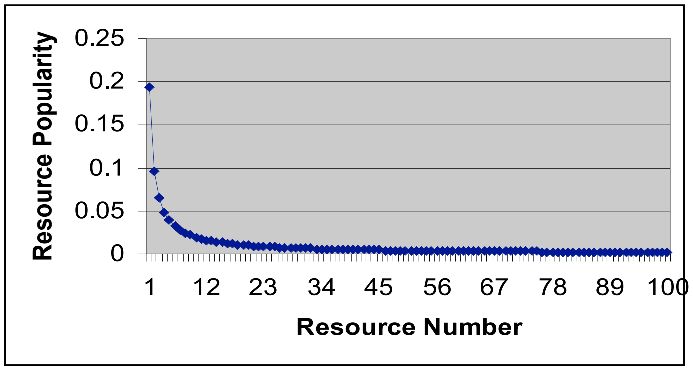


Figure 5-2. Zipf’s distibution (α=1)

Breslau et al. [64] studied traces from proxies at academic institutions, industry corporations and ISPs, and they made the following observations:

- For most traces, "the distribution of page requests generally follows a Zipf-like distribution where the relative probability of a request for the i'th most popular page is proportional to $1/i^{\alpha}$, with α typically taking on some value less than unity."
- "For traces from a homogeneous environment, α appears to be larger, and for traces from a more diversified user population, α appears to be smaller. Perhaps the biggest impact of α lies on the concentration of web requests to hot documents."

5.2 Experimental Setup

The experiments are executed on four machines of identical hardware and software configuration. Two kinds of machines are used for the experiments:

- Stationary machines: The hardware configuration of the stationary machine is Pentium® 4 CPU 3.19 GHz, 2.00 GB RAM. Stationary machines are connected to the network via 100Mbps Ethernet connections. The operating system on the stationary machines is Windows XP.
- Mobile devices: The mobile devices are Hp iPAX hx2700 PDAs, running Windows Mobile 2003 Second Edition and the .NET Compact Framework [65]. The hardware configuration of the PDAs is Intel PXA270 32 bit XSCALE RISC microprocessor with 128MB RAM. They are connected to the network via 54Mbps 802.11g network.

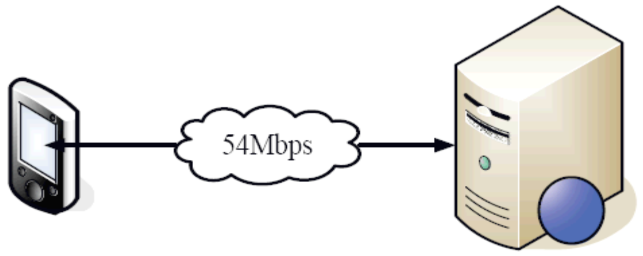


Figure 5-3. Setup of experiment system

Figure 5-3 shows the experimental setup for experiments in this chapter. The stationary machine acts as the WS provider. The WS has been implemented with Java 1.4.2 [66], Apache Tomcat 5.5.20 [67], and Apache Axis 1.2 [68]. The resources on the provider are 100 text files. The size of the files ranges from 2KB to 50 KB. The WS `read` and `write` operations are directly mapped to read/write operations of the files. There are 100 `read` and 100 `write` service operations.

The popularity of the files follows Zipf's distribution (Figure 5-2). Therefore, the popularity of a file depends on its sequence number. From the first to the one hundredth file, twenty files are [2KB, 10KB], thirty files are [10KB, 20KB], forty files are [20KB, 30KB] and ten files are [30KB, 50KB]. The smaller size files are more popular than the larger ones. The average size of the files is E(S) = 16KB.

The CSC is placed on the PDA that contains the client application. It is implemented with C#.NET as Microsoft Visual Studio provides a flexible way to develop applications for Windows Mobile powered devices. From the client's point of view, the CSC appears as a standard web proxy. The communication between the CSC and the client is over a local TCP connection through HTTP. The PSC is implemented with Java 1.4.2 also as a web proxy, which is placed on the same stationary machine of the WS provider. The communication between the CSC and the PSC is over a TCP connection via 54Mbps Wi-Fi. The CSC sends requests to the PSC that forwards them to the service provider. The two cache proxies are transparent to both clients and service providers.

5.3 Caching Performance

The experiments in this section are intended to determine the overhead and gains on the performance due to MDDC. Three workloads are used to evaluate the impact of the message size, the cache size, the replacement algorithm and levels of wireless connectivity on the performance of the Dual-Caching system.

5.3.1 Impact of Message Size

The first experiments are used to determine the impact of the message size on the performance of MDDC. Using a CSC and a PSC a sequence of unique `reads` is performed. The mean latency is calculated in terms of message size. Each experiment is performed without cache, with the PSC, with the CSC, and with the Dual-Caching.

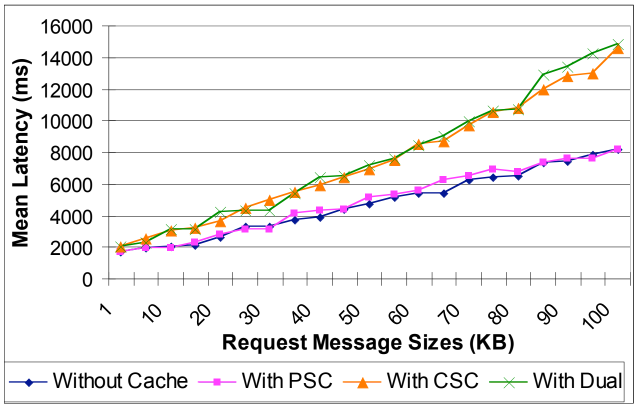


Figure 5-4. Impact of the Request Message Sizes

Figure 5-4 shows the mean latency as the request message size is changed from 1 to 100KB. This figure shows a linear increase of the mean latency as a function of the request message size.

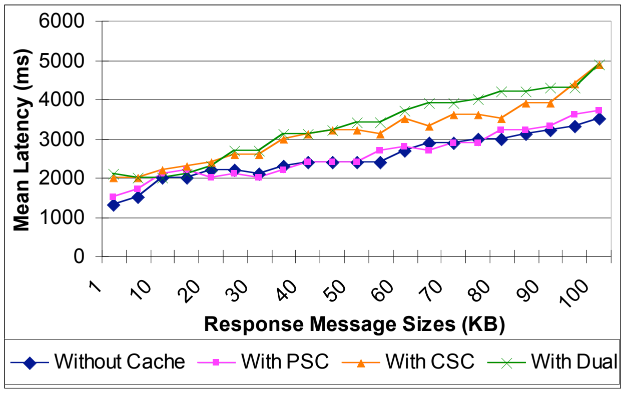


Figure 5-5. Impact of the Response Message Sizes

To determine the impact of the response message size, a set of `reads` is performed that result in different response message sizes. By changing the parameters, the response message size changes from 1 to 100KB. As shown in Figure 5-5, the increase of the response message size results in a linear increase of the mean latency.

Combining the two figures, we can see that the PSC adds a light overhead while the CSC adds a significant overhead. The overhead of MDDC is mainly caused by the overhead of the CSC. This is because the CSC resides on the PDA that has limited memory and processor power, while the PSC resides on a stationary machine that has powerful processor and large memory. Although both figures show a linear increase, the impact of the request message size is larger than that of the response message size. We can see from the figures, the mean latency of a 50KB request message with the Dual-Caching is 7200ms, while the mean latency of a 50KB response message is 3200ms. The results indicate that:

- The mean latency of a request message caused by the WS system is higher than that of a same size response message. E.g., the mean latency of a 50KB request message is 4700ms, while that of a 50KB response message is 2400ms.
- Because both the CSC & PSC analyze and use the request message as the hash key, they add more overhead when processing a larger request message.

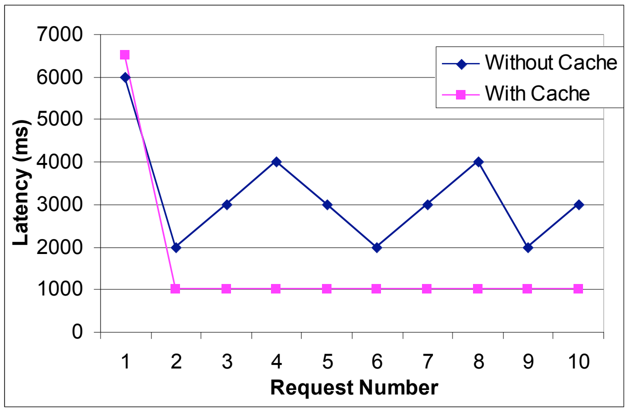


Figure 5-6. Latency Reduction

To determine the gains of using MDDC in a connected mode, a series of repeating reads is performed. The workload includes 10 identical `read` operations, whose request message size is 1KB and response message size is 50KB. Since identical `reads` are executed, it is possible to evaluate the gains due to the cache. The experiment is also conducted with and without cache.

As shown in Figure 5-6, MDDC reduces the latency dramatically since the sending of costly SOAP messages is avoided. Note that by using the cache, the service provider is subjected to a lighter load.

5.3.2 Evaluating Replacement Algorithms

Mobile devices have limited memory. This section investigates the impact of replacement algorithms on the performance of MDDC. The three workloads – browsing, shopping and ordering – differ in the ratios of `reads` (cacheable) and `writes` (uncacheable) that result in different invalidation ratios. The replacement algorithms evaluated here are LFU (Least Frequently Used), LRU (Least Recently Used) and SIZE (remove largest element in cache). The CSC & PSC are configured as the following:

- The CSC cache size changes from 0 to 800KB. The CSC stores the core-level service metadata (see Chapter 4 for detail), which indicates the cache-ability of each service.
- The PSC has infinite cache size. It stores the detailed-level service metadata, including dependency relationships among services. The PSC generated invalidation reports according to the invalidation relationship and piggybacks them as a part of the SOAP header in response messages to the CSC every 5 seconds.

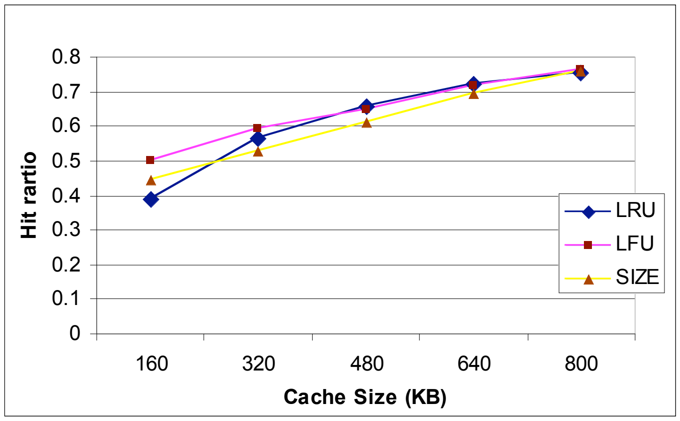


Figure 5-7. Cache-hit ratio for Browsing

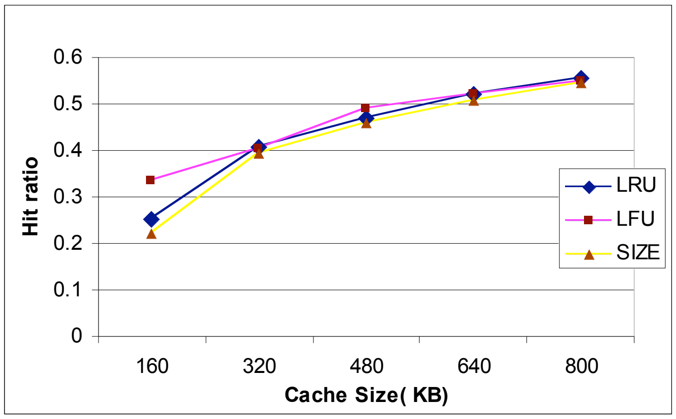


Figure 5-8. Cache-hit ratio for Shopping

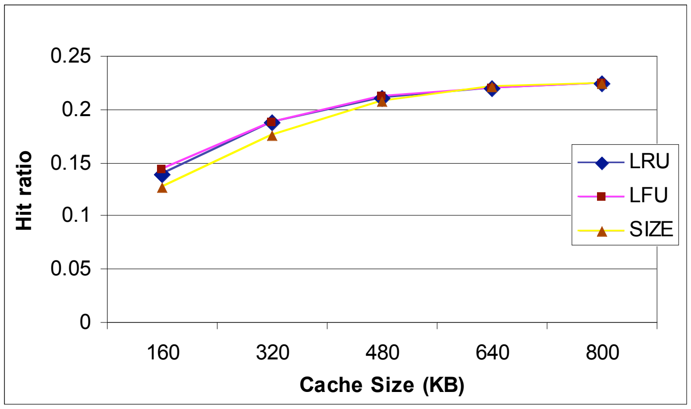


Figure 5-9. Cache-hit ratio for Ordering

Figure 5-7, 5-8 & 5-9 show the cache-hit ratios that are obtained for different replacement algorithms and cache sizes based on the three workloads. For each workload, LFU has the best performance, but the three figures only show slightly different performance among the replacement algorithms.

5.3.3 Impact of Cache Size

As mobile devices have limited processor power and memory, it is necessary to measure the cache management overhead of the CSC. Both the CSC & PSC use an in-memory hash table to cache the request/response message pairs. A hash table supports efficient lookup operation that associates keys with values. One advantage offered by the hash table is that the lookup time is independent of the number of items stored in it, i.e. O(1) (it takes a constant amount of time). Therefore, a large cache size for the CSC & PSC will not increase the lookup time. However, a large cache size can introduce a cache management overhead by increasing: (1) the time spent on garbage collection in the .NET framework; (2) the time spent on organizing and replacing cached items; (3) the time spent on invaliding obsolete cached items.

The performance parameter measured in this set of experiments is latency in terms of cache size. The PSC is configured to send metadata and invalidation reports to the CSC as described in Section 5.3.2. The `read/write` ratio of the three workloads is based on Table 5-1. The workloads are designed to simulate the worst situation in which the client always invokes different services and no cached items can be re-used. Every `read` operation has the same request (1KB) and response (30KB) message size. After a cold start period, the CSC has to evict a cached item for storing a new one.

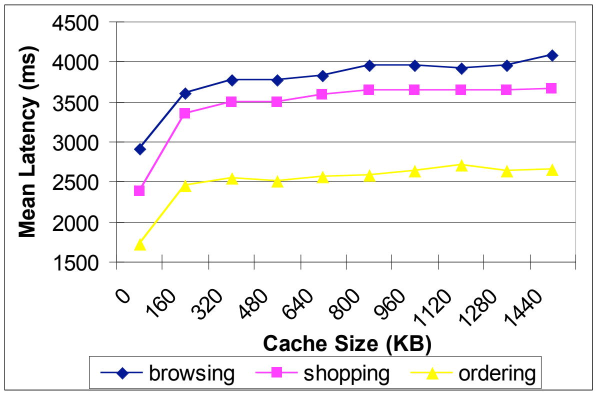


Figure 5-10. Mean latency in terms of cache size

Figure 5-10 shows the CSC management overhead as the mean latency of the three workloads. The mean latency is calculated after the cold start period. From 0 (without the CSC) to 160KB, the CSC adds about 700ms mean latency to each workload. However, as expected, the CSC only adds a slight overhead when the cache size increases from 160 to 1440KB.

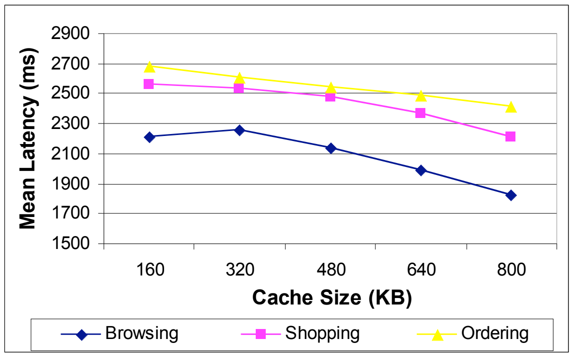


Figure 5-11. Mean latency in terms of cache size

Figure 5-11 shows the impact of cache size on the mean latency if LFU is used. The three workloads and the configuration of the CSC & PSC are described in Section 5.3.2. As expected, the cache size has the greatest impact for the workload primarily consisting of `reads`. Consequently, browsing (95% reads) benefits more than shopping (80% reads). Ordering which contains 50% reads benefits the least since many writes are not cacheable and invalidate cached reads. These figures indicate that a large cache size improves the performance impressively with adding a slight overhead.

5.3.4 Impact of Wireless Connectivity

The purpose of this experiment is to measure the performance of MDDC over unstable wireless connections. The wireless condition is simulated by a wireless simulator residing on the PDA. In the first experiments, the wireless condition is simulated as 50 seconds of connection separated by 5 seconds of disconnection.

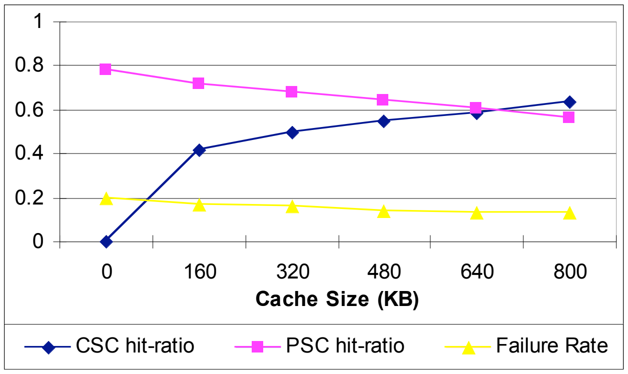


Figure 5-12. Performance for Browsing

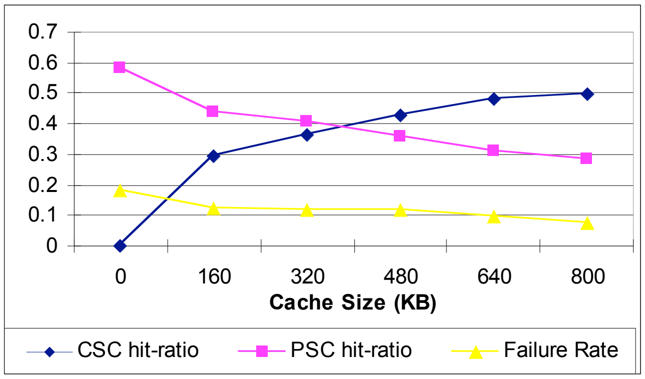


Figure 5-13. Performance for Shopping

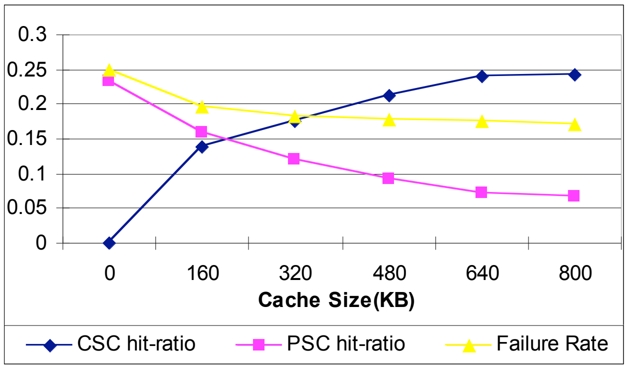


Figure 5-14. Performance for ordering

Figure 5-12, 5-13 and 5-14 show the cache-hit ratio of both the CSC & PSC and the failure rate of each workload. The failure rate is caused by the disconnections that interrupt the communication between the PDA and the service provider. Cache misses on the CSC can still have a cache hit on the PSC. As the CSC cache size is increased, more responses are served from the CSC and the failure rate is reduced. Meanwhile as more responses are served from the CSC (the CSC hit ratio increases) the PSC hit ratio decreases.

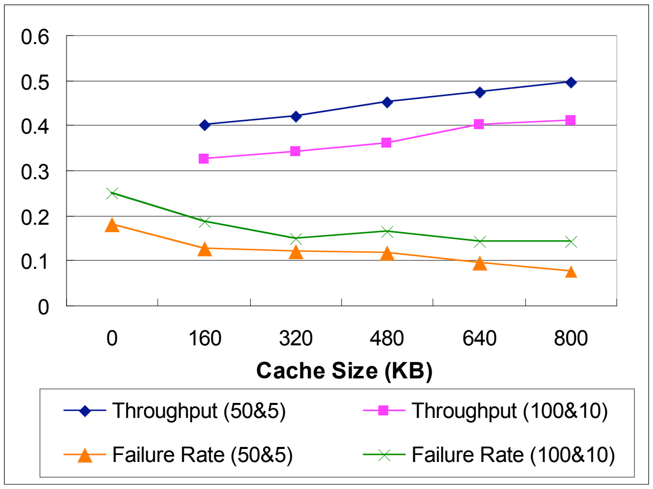


Figure 5-15. Throughput and failure rate

Another set of experiments is done under a different wireless condition, consisting 100 seconds of connection separated by 10 seconds of disconnection. Figure 5-15 shows the throughput and failure rate of the shopping workload under these two wireless conditions. The failure rates under both wireless conditions are only reduced slightly as the cache size increases. Moreover, the failure rate is higher under 100&10 than those under 50&5 and the throughput can not be improved by increasing the cache size.

5.4 Enhanced Caching Performance via Prefetching

The evaluation in the above sections is done by using the browsing, shopping and ordering workloads. As the results show the same trend for the three workloads, the experiments in the following sections only use the shopping workload.

This set of experiment aims to identify the costs and benefits of prefetching. Prefetching is performed by the CSC based on the client workflow model. Service metadata in the CSC includes a SOAP template for each cacheable service. The prefetching engine generates SOAP request messages by inserting the parameters to the SOAP template. This saves the time spending on SOAP encoding. The prefetching engine generates a thread for each request. As LFU may replace newly prefetched items, LRU is used for these experiments.

5.4.1 Overheads of Prefetching

This experiment determines the prefetching overhead of the CSC by measuring latency. In order to simulate the worst situation of prefetching, the workflow model of the prefetching engine is totally different from the one the client uses. In other words, the prefetching only adds overheads to MDDC as the accuracy of the client workflow model is zero. The step-ahead counter indicates how many steps the prefetching engine looks ahead and prefetches. For example, when the counter is set to three, the prefetching engine checks the next three operations and uses three threads to prefetch if needed.

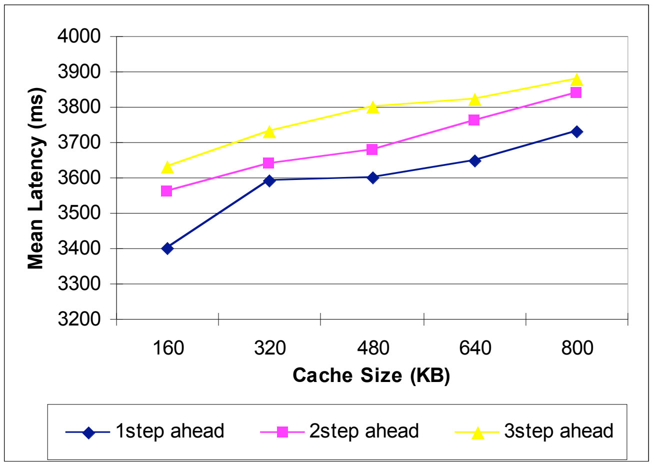


Figure 5-16. Prefetching overheads for shopping

Figure 5-16 shows a linear increase in the mean latency as a function of the cache size. The increase is less than 5% for each 160KB increase on the cache size. The prefetching causes a less than 3% increase in the mean latency as the step-ahead counter increases.

5.4.2 Prefetching with a Connected Mode

This section presents the enhanced cache performance via prefetching in a connected mode. The accuracy of the client workflow model is 100%, 80% & 60%. The percentage of accuracy indicates the percentage of correct operations the model has. The CSC size is 800KB and the replacement algorithm is LRU.

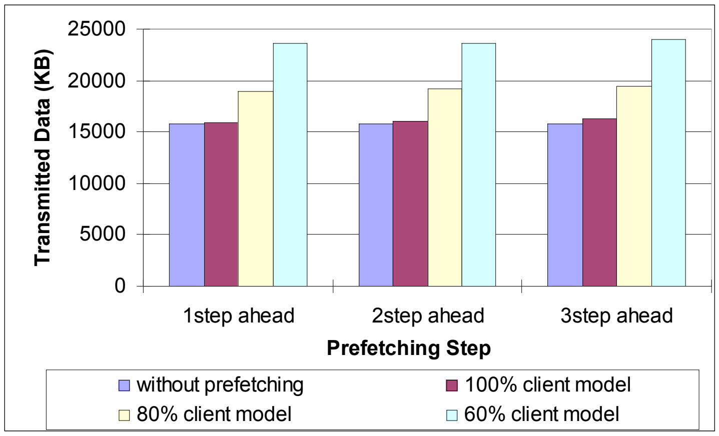


Figure 5-17. Transmitted data bytes

Figure 5-17 shows the transmitted data bytes of each workflow model, which represents the bandwidth consumption overhead of prefetching. A 100% accurate client workflow model adds almost no bandwidth consumption overhead. As the accuracy decreases, the bandwidth consumption overhead is increased because some prefetched items are never used.

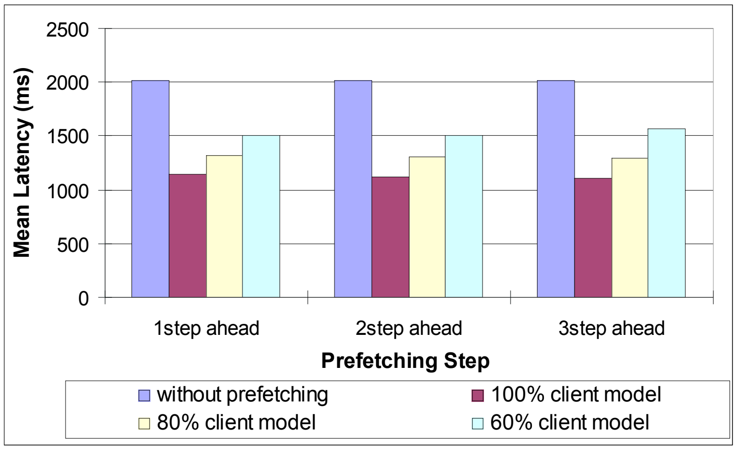


Figure 5-18. Mean latency

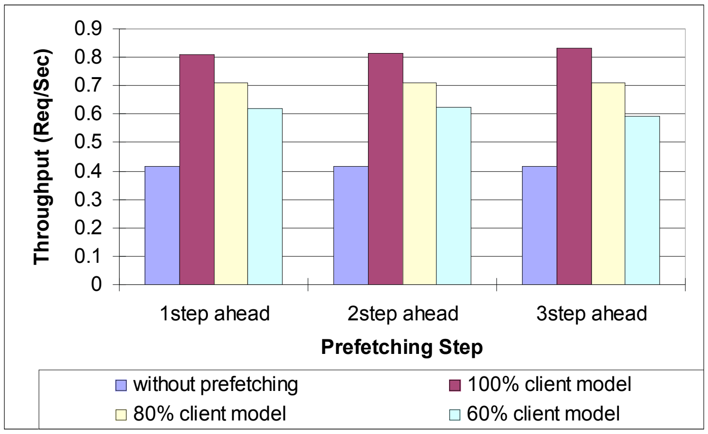


Figure 5-19. Throughput

Figure 5-18 & 5-19 show that the mean latency and the throughput have been dramatically improved by prefetching. As the model accuracy decreases, the improvement in performance by prefetching decreases. However, even with a 60% accurate client workflow model, prefetching still reduces the mean latency by 25% and increases the throughput by 48%. These two figures also show that increasing the step-ahead counter to prefetch does not improve the performance much in a connected mode. Moreover, Figure 5-18 shows that the mean latency for 3-steps ahead prefetching is slightly higher than that of 2-step ahead prefetching. This result indicates that, with a stable wireless connection, 1-step or 2-steps ahead prefetching is a better strategy for good caching performance.

5.4.3 Prefetching with a Disconnected Mode

As discussed in Section 5.3.4, MDDC exhibits a limited improvement in performance when an increased cache size issued for long periods of disconnection. This section aims to verify the improvement in performance by prefetching under different wireless conditions. The experiments are done under different wireless connections: 50&5 means 50 seconds of connection separated by 5 seconds disconnection; 100&10 means 100 seconds connection separated by 10 seconds disconnection; Mixed means a mixed connection of the two connections above.

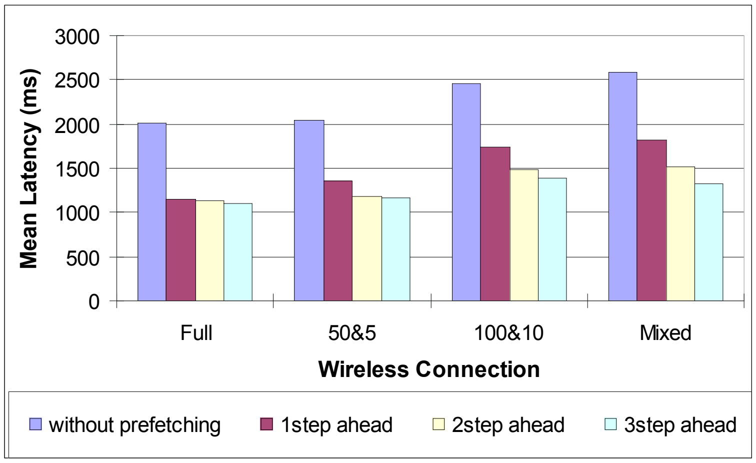


Figure 5-20. Mean latency with the 100% accurate client workflow model

Figure 5-20 shows the mean latency under different wireless conditions. From this figure we can observe a clear improvement in performance for increases in prefetching look-ahead steps when the periods of wireless disconnection are longer. For example, in the case of the 100&10 wireless condition, using 3-steps ahead prefetching reduces the mean latency by 214ms over 2-steps ahead prefetching. Meanwhile, for the 50&5 wireless condition, the mean latency is only reduced by 116ms.

5.5 Performance with Different Accuracy Models

This set of experiments measures the impact of different model accuracy on the performance of MDDC. The CSC cache is set to 800KB and uses the LRU replacement algorithm.

The 100% accurate client workflow model means that the prefetching engine can predict exactly the next step the client will take. The 80% accurate client workflow model means 80% of the predictions are correct, but other 20% are wrong.

The wireless monitor generates different periods of connection and disconnection. The prefetching engine sets its step ahead counter based on the wireless condition. When the wireless condition is a stable connection, the step ahead counter is set to two. When a disconnection is detected, the prefetching engine sets the counter based on the length of the disconnection.

The 100% accurate wireless model means the prefetching engine knows when disconnections will happen and how long they will last. The 80% accurate wireless model means that 80% disconnection can be detected.

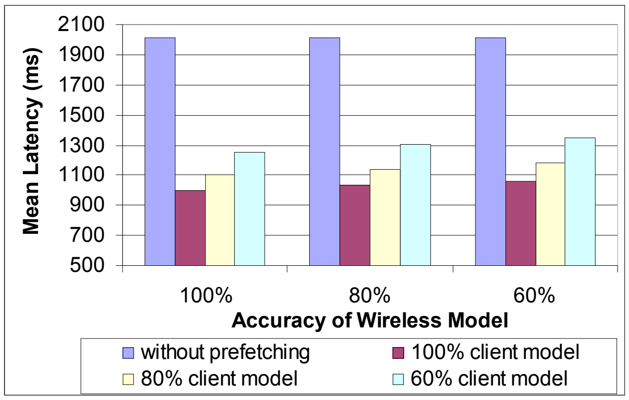


Figure 5-21. Mean latency with 100% accurate service metadata

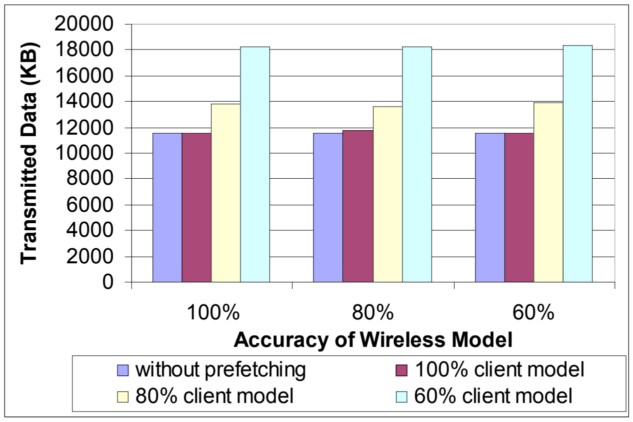


Figure 5-22. Transmitted data with 100% accurate service metadata

Figure 5-21 & 5-22 show costs and benefits of MDDC with different client workflow and wireless accuracy models. In this experiment, MDDC has 100% accurate service metadata. As expected, when MDDC has a 100% accurate client workflow model and wireless model, it has the best performance. It improves the performance by reducing 50% mean latency, while the increased cost (bandwidth consumption) is almost 0%. When the wireless model is 100% accurate, the 80% accurate client workflow model reduces the mean latency by 12% and the bandwidth consumption by 24% when compared to the 60% accurate client workflow model.

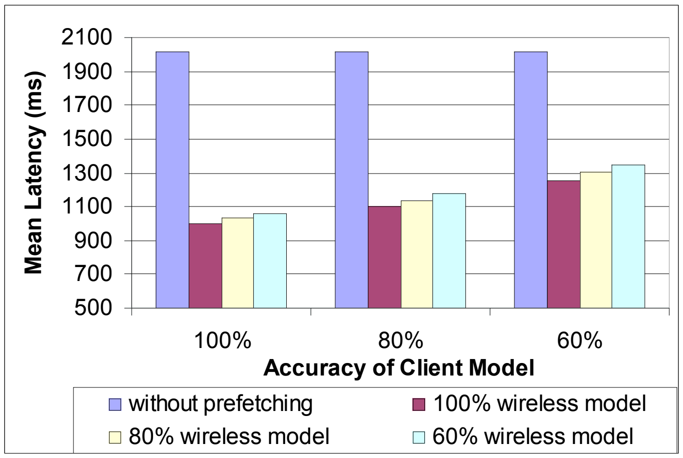


Figure 5-23. Mean latency with 100% accurate service metadata

Figure 5-23 shows the performance improvement from another point of view. When the accuracy of the client workflow model is the same, the accuracy increase of the wireless model does not reduce the mean latency as much as the client workflow model does. For example, when the client workflow model is 100% accurate, the 80% accurate wireless task model reduced the mean latency by less than 4% compared to the 60% accurate wireless model. This might be caused by the `write` operations. Even the prefetching engine knows when a disconnection will occur and how long it will last, it cannot prefetch `write` operations.

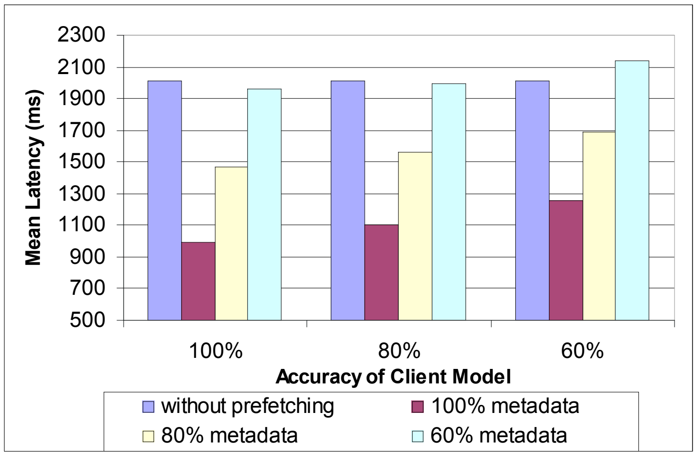


Figure 5-24. Mean latency with 100% accurate wireless model

Figure 5-24 shows the impact of the service metadata accuracy. The experiments are done under a 100% accurate wireless model. When the accuracy of service metadata is reduced from 100% to 80%, the mean latency increases by 78%. Compared to other figures (5-20 & 5-21), we can observe that service metadata has the greatest impact on the performance of MDDC.

5.6 Conclusions

MDDC adds overheads to both the client-side and the provider-side machines. The overhead added to the client-side machine is higher since the CSC is running on a resource limited PDA. The overhead varies according to the message size and the cache size. A larger message adds a greater overhead to the system. However, a large cache size only adds a slight overhead. The improvement in the performance of MDDC depends on the percentage of the cacheable operation in the workload, the replacement algorithms, the cache size, and the levels of wireless connectivity. The higher the percentage of the cacheable operation in the workload and the larger the cache size is, the greater improvement in performance MDDC achieves. MDDC performs better under shorter disconnections than it does under longer disconnections. Moreover, this limitation cannot be covered by increasing the cache size.

Prefetching adds a slight overhead to the system, which varies according to the cache size and the number of prefetching steps. Prefetching improves the performance dramatically under both stable connectivity and unstable connectivity. The improvement in the performance depends on the accuracy of the client workflow model, the wireless network model and service metadata. The more accurate the models are, MDDC performs better and the less overhead is added. Among the models, service metadata is the most important model and the performance of MDDC depends heavily upon it. The second important model is the client workflow model because MDDC can not perform prefetching without it. Finally the wireless network model is the least important model.

CHAPTER 6

EVALUATION WITH A BOOKSTORE SCENARIO

Following the evaluation of MDDC with abstract workloads, this chapter presents a set of experimental results based on a bookstore scenario. This set of experiments is aimed at evaluating the MDDC framework with a real world scenario.

6.1 Scenario

X is an undergraduate student at the University of Saskatchewan. It is the beginning of a new term. He has registered for several courses and spends most of his day on campus. One of his tasks is to buy textbooks and reference books for his courses. Instead of going to the campus bookstore, which is very crowded and where the books are new and expensive, he goes to amazon.com to find good deals.

X has a PDA and uses Wi-Fi to access the internet. The wireless network is provided by the UofS. Wireless Access Points are installed in a number of public locations on campus, providing wireless network access to members of the campus community. Many buildings on campus (but not all) have wireless access coverage.

As X is busy walking between classrooms located in different buildings, he does not have a long period of time in which to browse the website. Therefore, he takes advantage of the Amazon WS to find books and make orders. He wrote a short program which takes keywords as parameters to fetch book information to his PDA, and then he can browse the information and make an order whenever he has time. He wants to do it before the class begins or during his lunch break. Due to the coverage changes over the campus wireless network, he may experience variable wireless connectivity:

- Strong connectivity: When his classroom is in a building with a wireless access point, he will experience strong connectivity; and
- Weak connectivity: When he is sitting on the grass beside a building with a wireless access point, he may experience weak connectivity; and
- Null connectivity: When he is having lunch off campus, he can not access the wireless network.

6.1 Experimental Setup

The hardware configuration and the OS of the stationary machines and mobile devices are the same as described in Section 5.2. The bookstore WS resides on a stationary machine, while the client application resides on a PDA.

6.1.1 Bookstore Web Services

In order to simulate the scenario described in Section 6.1, a bookstore WS framework has been implemented based on the TPC-App [61]. "TPC-App is an application server and WS benchmark. The workload is performed in a managed environment that simulates the activities of a business-to-business transaction application server operating in a 24/7 environment". Table 6-1 shows the eight operations exposed by the bookstore WS. It also shows the attributes of each operation.

Table 6-1. Bookstore WS

Operation Name	Cacheability	Invalidation
NewCustomer	N	
CreateOrder	N	ProductDetails, OrderStatus
OrderStatus	Y	
NewProducts	Y	
BestSellers	Y	
ProductDetails	Y	
ProductsDetails	Y	
ChangeItems	N	ProductDetails, NewProducts

The bookstore WS framework has been implemented using JSE 1.4.2, Apache Tomcat 5.5.20, and Apache Axis 1.2. The application server is Tomcat 5.5.20 running on JSE 1.4.2. Apache Axis 1.2 is chosen as the base on which to implement Java WS. MySQL 4.1 [69] is used as the back-end database server. The database contains all the information operated on by the bookstore services. It has five tables: *author, customer, cust_order, order_details,* and *books.* The size of the database is 58,800 KB. The *books* table contains 11,342 records which were selected from www.amazon.com.

The bookstore services and MySQL database server reside on the same stationary machine.

6.1.1.1 Client Process Workflow

Figure 6-1 shows the client process workflow which follows the scenario described in Section 6.1. First, the client inputs keywords of book titles. Then, the client software invokes the *NewProducts* operation to find recently published books. After getting the service response (an array of ISBNs), it invokes the *ProductsDetails* operation to get detailed information on each book. We can see from the workflow, it only fetches information one book at a time. A longer message will result in longer latency and higher failure rate (see Figures 5-4 & 5-5). The transmission may be interrupted during any period when the wireless connection is not stable (Section 2.2).

The client will review the books one by one. If the client finds one he needs, he will make an order with the *CreateOrder* operation. When the while-loop finishes, the client checks his order status with the *OrderStatus* operation. If he requires more books, the *BestSeller* operation will be invoked to retrieve another 50 books.

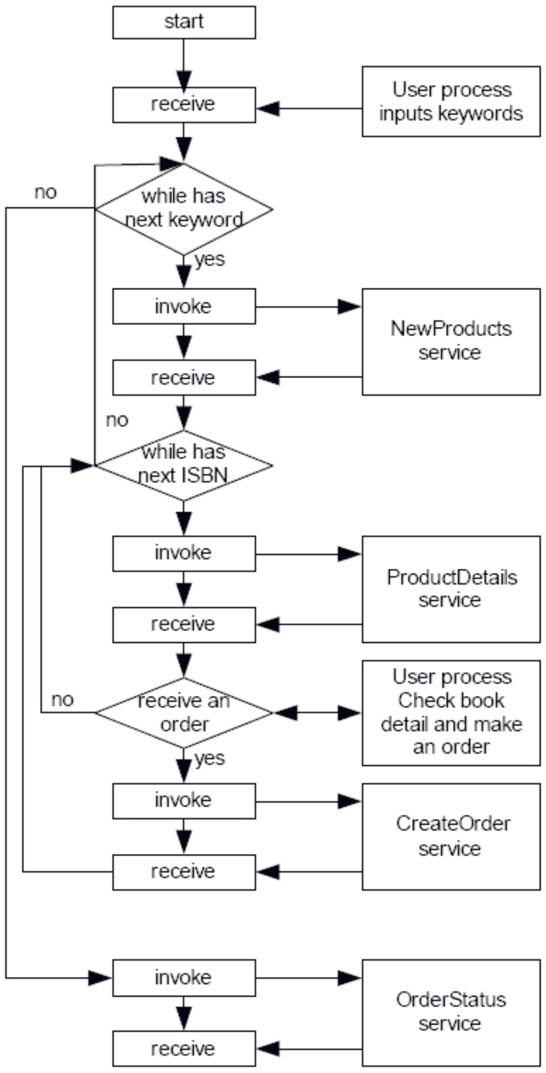


Figure 6-1. The client process logic

6.1.1.2 The Dual-Caching System Setup

From the flowchart in Figure 6-1, we can see *ProductDetails* is the most frequently invoked operation (this is also true according to the TPC-App benchmark). The average output message size of this operation is 5KB (including the book description and customer reviews). The CSC size is set to 250KB and it stores the core-level service metadata (Section 4.2). The CSC prefetching engine obtains the client process logic by analysing the workflow specification. When the prefetching engine captures a *NewProducts* or *BestSellers* return message, it decodes the SOAP message to get the ISBN list. Then it prefetches the book information for each ISBN by calling *ProductDetails* until it reaches the end of the list or the client jumps out of the while-loop.

The PSC has an infinite cache size and detailed service metadata. In addition to caching request/response message pairs, the PSC generates invalidation reports based on the service metadata and piggybacks these reports to the CSC every 5 seconds.

6.2 Experimental Results

This section presents the results for evaluating the MDDC framework based on the bookstore scenario.

6.2.1 Performance with Strong Connectivity

This set of experiments evaluates the performance of MDDC with strong connectivity. In these experiments, the client gives seven keywords, browses detailed information of every book, and makes orders. The interval time between each request is 200ms. This nomadic client is assumed to be the only client consuming services using MDDC. Figure 6-2 shows the mean latency under different conditions – without cache, with the PSC, with the CSC, with the Dual-Caching, and with one-step ahead prefetching.

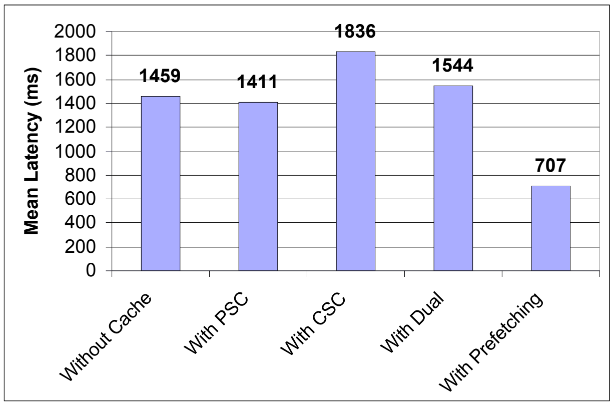


Figure 6-2. Mean latency

First the base system performance, i.e. the latency without any cache, is measured. Figure 6-2 shows the mean latency is 1459ms. Comparing it to the value of *With CSC*, we can see that the CSC imposes an overhead on the system by increasing the mean latency to 1836ms. This is caused by the characteristic of the workload and the limitation of the local storage. The client keeps looking for different books in each request which results in a 0% cache-hit ratio. The mean latency is reduced slightly with the PSC and with the Dual-caching. This improvement is achieved through the 3% cache-hit ratio of the PSC as the PSC has infinite cache size and the *NewProducts* operation and the *BestSeller* operation return the same ISBNs in some cases. The last column shows an impressive improvement in the performance by utilizing the one-step ahead prefetching of MDDC. Compared to the base system performance, the mean latency is reduced by 52%. Figure 6-2 highlights the importance of the prefetching strategy based on the client workflow model in the bookstore scenario. The characteristics of this bookstore workload are different from the ones used in Chapter 5 in that the nomadic client can not take advantage of previous requests. Caching itself can not improve the performance and adds an overhead. Prefetching leads to a noticeable speedup of service requests from the client's perspective.

6.2.2 Performance with Weak Connectivity

This set of experiments measures the impact of an unstable wireless network on the performance of the MDDC system. The wireless monitor generates different periods of connection and disconnection. In this set of experiments, the wireless connectivity is set to 100 seconds of connection separated by 20 seconds & 40 seconds disconnections. During a disconnection, the client will keep sending the same request until it receives a response. The inter-arrival time of each request during disconnection is 1200ms.

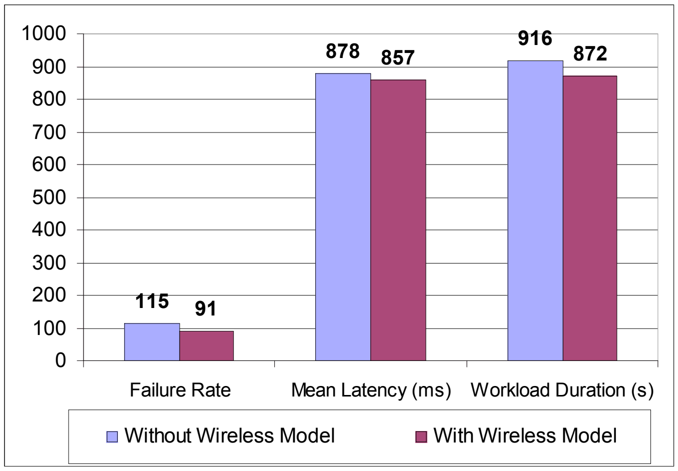


Figure 6-3. Performance with weak & null connectivity

One-step ahead prefetching produces an impressive reduction in client latency as shown in Figure 6-2. Therefore, the first experiment assumes the CSC has no knowledge of the wireless connectivity status and measures the performance of one-step ahead prefetching with weak & null connectivity. The second experiment assumes the CSC has a wireless network model which predicts when a disconnection will occur and how long it will last. The prefetching engine resets its step ahead counter based on the information.

When the wireless connection is stable, the counter is set to one. The prefetching engine retrieves the service at the first possible moment during client idle time. When a disconnection is detected, the counter is set based on how long the disconnection is expected to last. The services are prefetched and hoarded just before the disconnection is expected to occur. Figure 6-3 shows the improvement in performance with the wireless network model. The failure rate is reduced by 21%, and the whole workload duration and mean latency are improved by 2.3% and 5% respectively.

6.3 Experiments with a Homogenous User Population

The experiments of previous sections assume the nomadic client is the only client using MDDC and the bookstore services. This set of experiments investigates the impact of a user population and the granularity of service metadata on the performance of the MDDC system.

6.3.1 Experimental Setup

Figure 6-4 shows the setup of the evaluation system. The mobile devices and the web application server have the same hardware configuration and operating system as described in Section 6.2. The stationary machine acts as the web application server. There are two types of workloads in this set of experiments: the single-client workload and the multi-client workload. The first workload is the same as was described in Section 6.2.1.1 which simulates a single bookstore client with a mobile device. The second workload simulates multi-client behaviors with mobile devices, which also runs on a PDA.

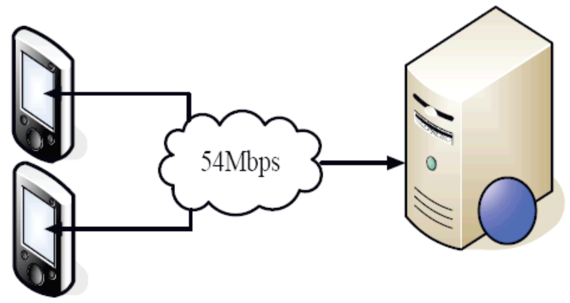


Figure 6-4. Setup of the evaluation system

Table 6-2 shows the distribution of each WS operation in the second workload. The multiple clients are not assumed to be pure TPC-App or pure TPC-W clients. Therefore, this workload was created by combining the TPC-App & TPC-W benchmarks.

Table 6-2. Web service operation mix

Web service operation	Distribution
New Customer	1%
Create Order	20%
Order Status	5%
New Product	7%
Best Sellers	7%
Product Detail	56%
Change Item	4%

The multi-client workload has 2500 WS operations. The inter-arrival time follows an exponential distribution with a mean of 20ms. After 1000 operations, the single-client workload begins. The book information in the MySQL database is categorized by book title keywords, which will be used in the *BestSellers* & *NewProduct* operations. The popularity distribution of the keywords follows a Zipf's-like distribution with α=1.0. The popularity distribution for the books also follows a Zipf's-like distribution with α=1.0.

6.3.2 Performance with Strong Connectivity

The granularity of the invalidation relationship has two levels:

- Coarse-grained: Service name + Operation name.
- Fine-grained: Service name + Operation name + Parameters.

As shown in Table 6-1, the *CreateOrder* operation invalidates the *ProductDetails* and *OrderStatus* operations. As *ProductDetails* is the most frequently called operation in both workloads, the granularity of the invalidation relationship between *CreateOrder* and *ProductDetails* is crucial. Two levels of service metadata granularity are defined in this scenario:

- Coarse-grained: the *CreateOrder* operation invalidates all cached *ProductDetails* items in both the PSC & CSC.
- Fine-grained: the PSC decodes the input SOAP message of the *CreateOrder* operation to get the parameter (the ISBN). Then it only invalidates the cached *ProductDetails* items with the same parameter.

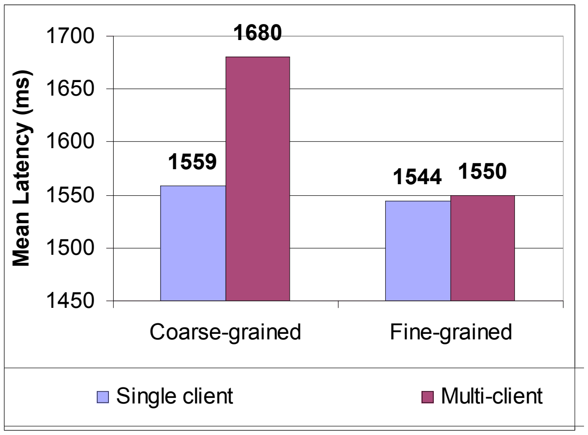


Figure 6-5. Mean latency

Figure 6-5 shows the mean latency observed by the bookstore client. Compared to the single client workload, the multi-client workload increases the mean latency with both the coarse-grained and the fine-grained service metadata. This is caused by the load on the server and the PSC added by the multi-client workload. When the single client workload is the only workload, the fine-grained service metadata improves the performance slightly – only from 1559 to 1544ms. This is because the single client workload only has 4% *CreateOrder* Operations. Meanwhile, as shown in Table 6-2, the multi-client workload has 20% *CreateOrder* operations and the improvement by the fine-grained service metadata is noticeable – the mean latency has been reduced by 8%. This improvement is achieved through cache-hits in the PSC. As mentioned in Section 6.3.1, if the bookstore client keeps looking at different books, the result is a 0% cache-hit ratio. Therefore, the granularity of service metadata has no impact on the CSC hit ratio in this scenario.

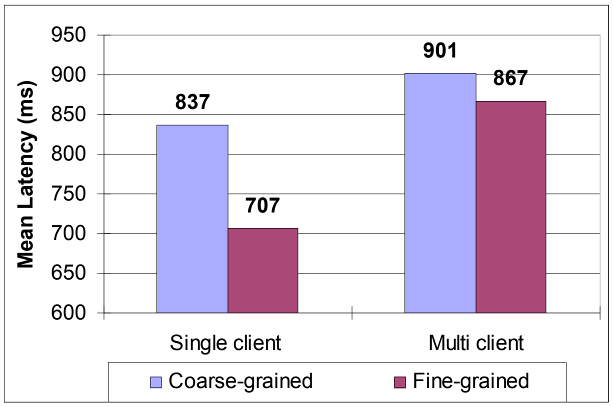


Figure 6-6. Mean latency

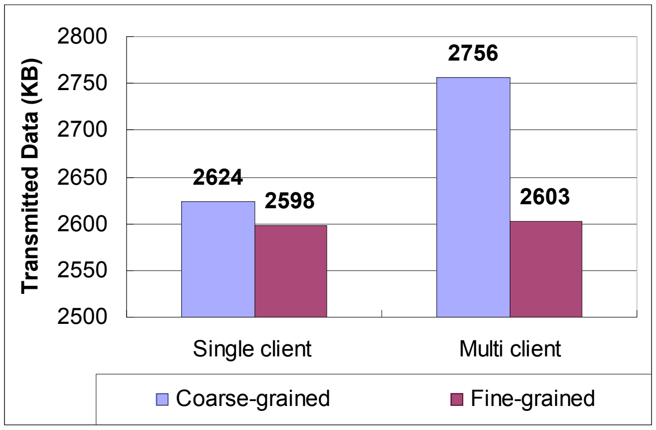


Figure 6-7. Transmitted data bytes

Figure 6-6 & 6-7 show the impact of user population and service metadata granularity on the performance of MDDC with the prefetching technique. Unlike the scenario in Figure 6-5, the CSC prefetches and hoards service messages based on the client workflow model. Prefetched items might be invalidated before they are accessed, which results in latency and bandwidth consumption overheads. Compared to the fine-grained service metadata, the coarse-grained service metadata increases the mean latency by 18% for the single client workload and by 4% for the multi-client workload. Unlike mean latency, the transmitted data bytes are increased by 6% for the multi-client workload and only 1% for the single client workload. As the multi-client workload has 20% more *CreateOrder* operations, the results indicates that the fine-grained service metadata reduces the invalidation rate on the CSC but adds a latency overhead for analyzing and generating detailed invalidation reports.

6.3.3 Performance with Weak Connectivity

This set of experiments evaluates the MDDC system in disconnected mode. MDDC has four different configurations: with/out the wireless model and coarse-/fine-grained service metadata. The experiments are conducted with the multi-client workload by combining the four configurations. The prefetching strategy is the same as described in Section 6.3.2.

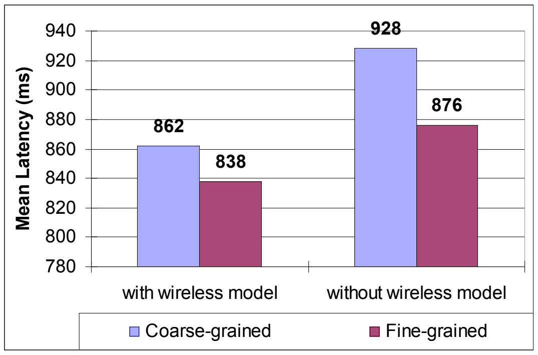


Figure 6-8. Mean latency

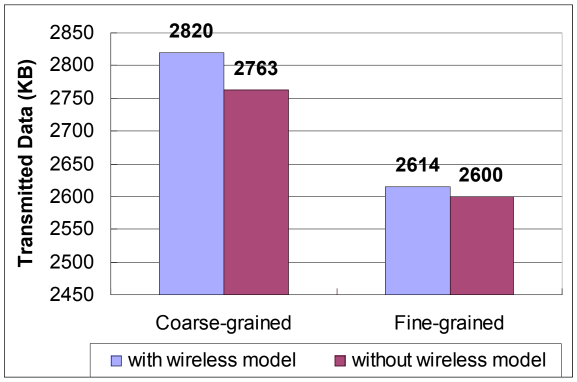


Figure 6-9. Transmitted data bytes

Figure 6-8 & 6-9 show the performance of the four different configurations. As expected, MDDC has the lowest mean latency when using the fine-grained service metadata and the wireless network model. Figure 6-9 shows the cost of using the wireless network model is the increase in bandwidth consumption.

6.4 Conclusions

The evaluation of MDDC in this chapter is based on a bookstore scenario. The single client workload is different from the workloads in Chapter 5 in that the client continues to look for different books in each service request. Caching itself can not improve the performance but adds an overhead because the nomadic client can not take advantage of the previous requests in the CSC. Prefetching of the CSC leads to a noticeable speedup in service requests from the perspective of the client. Therefore, the client workflow model for the MDDC prefetching strategy is crucial in this scenario. When the wireless connection is unstable, a wireless network model improves the MDDC performance.

The benefit of the homogenous user population is that it increases the PSC hit ratio, while the cost is an increase of the invalidation rate. In this situation, the granularity of service metadata has an impact on the performance of MDDC. Compared to the coarse-grained service metadata, the fine-grained service metadata improves the MDDC performance by reducing the mean latency and bandwidth consumption. When the wireless connection is unstable, the best performance is achieved by prefetching with a wireless network model and fine-grained service metadata.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

The Service-Oriented Architecture (SOA) is emerging as a new paradigm for building large, open, loosely-coupled systems. Unlike its predecessors (e.g. CORBA), SOA enables organizations to expose their legacy system in a way that supports the easy and fast creation of new services. But the widespread use of SOA raises questions for users of mobile devices such as laptops or PDAs; how and if they can participate in SOA. Unlike their "wired" counterparts (desktop computers and servers) they rely on a wireless network that is characterized by low bandwidth and unreliable connectivity. To overcome the problems of wireless connectivity, it is necessary to use caching and prefetching techniques.

This research focuses on the challenges of enabling the hosting of service consumers on mobile devices and introduces a Model-Driven Dual-Caching (MDDC) approach to overcome the problems which arise from the temporarily loss of connectivity and fluctuations in bandwidth. Using MDDC it becomes possible to handle the short periods of lost connectivity which occur during the transmission of requests and responses. In order to cache and prefetch services for nomadic clients in SOA, this research identifies important aspects of modeling the service, the client, and the wireless network on issues for caching and prefetching. By combining the three models, MDDC ensures the nomadic clients are able to seamlessly access the services and data offered in SOA environments.

The MDDC framework has been implemented and evaluated with synthetic workloads and a bookstore scenario. The experiments with the synthetic workloads show that:

- The improvement in the performance due to MDDC depends on the nature of the workload, the caching configuration, and the wireless network condition. The reduction in latency and network load is noticeable higher for workloads that contain significantly more `read`s than `write`s.
- Prefetching adds a slight overhead to the system, which varies according to the cache size and prefetching steps. Prefetching improves the performance dramatically under both strong and weak connectivity.
- The improvement in the performance depends on the accuracy of the three models (service metadata, client workflow model, and wireless network model). Among the three models, service metadata is the most important model as it provides fundamental information for caching services. The second most important model is the client workflow model as MDDC cannot prefetch services if it cannot predict the client's next steps. Finally, the wireless network model allows MDDC to be adaptive in an environment of changing wireless connectivity.

The workload in the bookstore scenario differs from the synthetic workloads in that the client keeps requesting different books in each service request. Caching itself can not improve the performance and adds overheads because the nomadic client can not take advantage of previous requests in the CSC. The experiments using the bookstore scenario highlight the importance of the client workflow model and the wireless network model, on which prefetching leads to a noticeable speedup of service requests under both strong connectivity and weak connectivity.

7.2 Future Work

Future work will focus on the extension and implementation of these models:

- Service metadata: The service metadata currently being used in the MDDC framework relies primarily on user-input specifications. Section 3.4 discusses the limitations of this approach and the possibility of generating service metadata for caching using semantic WS. The next step is to develop an automatic metadata detection system that can identify caching-related attributes from the information offered by the Semantic WS. This system could improve the accuracy of the service metadata and reduce the need for user intervention in cache management.
- Client model: Extending OWL to better model consumer activity, since the current OWL & OWL-S do not provide mechanisms for describing the behavior of consumers. One possible approach would be the introduction of OWL-C (OWL-Consumer).
- Wireless network model: Currently, the wireless network model is simulated by a wireless network simulator, which only simulates connections & disconnections in terms of time. While the statistics of past movements may serve as a good start, it seems more promising to incorporate mechanisms capable of predicting the location, movement and consequently the availability of connectivity. If the mobile device is equipped with a GPS receiver, the client's location, speed and direction could be identified. Moreover, the environment can be determined by using a GPS moving

map, e.g. locating base stations and nearby tall buildings. In the future, a wireless network monitor system may be developed by using GPS-relating applications.

- Another model that has not been discussed is the model of the cache itself (a semantic cache). As Table 3-2 shows, the majority of the proposed WS caching systems are based on original messages exchanged between the services. They use the request message as the key entry and store the response message in a hash table. An advantage of the source level caching is that there is no need for extra processing of cached items. However, it also has several disadvantages. First, SOAP messages are notorious for their unexpectedly long size. Mobile devices usually have a limited memory size, so some researchers doubt the efficiency of SOAP caching system for nomadic clients. Second, SOAP messages have various formats. For example, the SOAP messages generated by .NET are different from those generated by AXIS, which can result in cache-misses. Finally, as the number of caching entries increases, the performance of cache-hits may decrease. A semantic cache maintains a semantic description of the data in its cache. Semantic caching has been applied for Location-Based Services (LBS) in mobile computing. The idea of a semantic cache is consistent with that of the Semantic WS. One possible approach applying semantic caching to MDDC is to use OWL/OWL-S to include a semantic description for cached item in MDDC.

APPENDIX A

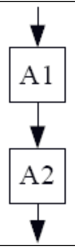
SERVICE METADATA DESCRIPTION WITH RDF/RDFS

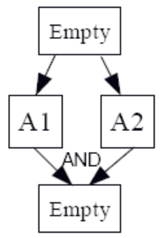
```
<?xml version="1.0"?>
<rdf:RDF
        xmlns:rss="http://purl.org/rss/1.0/"
        xmlns="http://a.com/servicemetadata#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
        xmlns:xsd=" http://www.w3.org/2001/XMLSchema#"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:base="http://a.com/servicemetadata">
<rdf:Property rdf:ID="Cachebility">
        <rdfs:range rdf:resource="&xsd;boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="Replay">
        <rdfs:range rdf:resource="&xsd;boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="DefaultReturnValue">
        <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="CommunicationStyle">
        <rdfs:range rdf:resource="&xsd;boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="MeanSizeOfResponse">
        <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Property rdf:ID="MeanBandwidthCost">
        <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Property rdf:ID="MeanUpdateRate">
        <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Class rdf:ID="InvalidationDependency">
<rdf:Class rdf:ID="PredictionDependency">
<rdf:Property rdf:ID="InvalidationParameter">
        <rdf:domain rdf:resource="# InvalidationDependency">
        <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="PredictionWeight">
        <rdf:domain rdf:resource="# PredictionDependency">
        <rdfs:range rdf:resource="&xsd;double"/>
</rdf:Property>
```

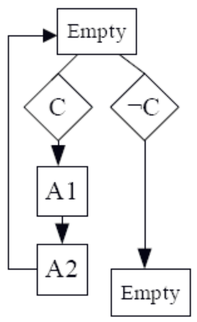
APPENDIX B

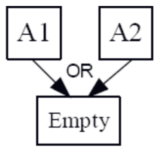
BPEL STRUCTURED ACTIVITIES

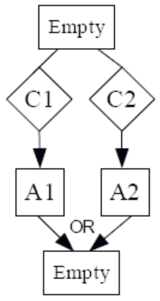
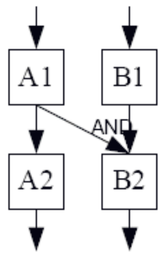
Activity	Sample Code	Description	Graph

Sequence	`<sequence…>`
	`    <…activityA1…>`
	`    <…activityA2…>`
	`</sequence…>`
		A set of activities performed sequentially	

Flow	`<flow…>`
	`    <…activityA1…>`
	`    <…activityA2…>`
	`</flow…>`
		A set of activities performed in parallel	

While	`<while condition = "bool-expr" …>`
	`    <…activityA1…>`
	`    <…activityA2…>`
	`</while>`
		Iterate execution of activities until the given Boolean condition is violated	

Pick	`<pick>`
	`    <onMessage m1>`
	`        <…activityA1…>`
	`    </onMessage m1>`
	`    <onMessage m2>`
	`        <…activityA2…>`
	`    </onMessage m2>`
	`</pick>`
		A set of events is being executed. Selects the activity corresponding to the first event finished.	

Switch	<switch condition_name> <case condition = cond1> <..activityA1..> </case> <case condition = cond2> <..activityA2..> </case> </switch>	Support conditional behaviors as case, otherwise branches	
Link	<flow…> <links> <link name=”L”/> </link> <sequence> <…activityA1…> <source linkName=”L”/> <…activityA2…> </sequence> <sequence> <…activityB1…> <…activityB2…> <target linkName=”L”/> </sequence> </flow…>	Defines a control dependency between a source activity and a target	

REFERENCES

[1] D. Box. Four tenets of service orientation. [Online]. *2007(06/21),* Available: http://msdn.microsoft.com/msdnmag/issues/04/01/Indigo/default.aspx

[2] J. Chatarji. Introduction to service oriented architecture (SOA). [Online]. *2007(06/21),* Available: http://www.devshed.com/c/a/Web-Services/Introduction-to-Service-Oriented-Architecture-SOA/

[3] R. W. Schulte and Y. V. Natis. (1996, 12 April). Introduction to service oriented architecture. [Online]. 2007*(06/21),* Available: http://www.gartner.com/DisplayDocument?doc_cd=114295

[4] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris and D. Orchard. Web services architecture. [Online]. *2007(06/21),* Available: http://www.w3.org/TR/ws-arch/

[5] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau. Extensible markup language (XML) 1.0. [Online]. *2007(06/21),* Available: http://www.w3.org/TR/xml/

[6] SOAP version 1.2. [Online]. *2007(06/21),* Available: http://www.w3.org/TR/soap/

[7] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana. Web services description language (WSDL) 1.1. [Online]. *2007(06/21),* Available: http://www.w3.org/TR/wsdl

[8] UDDI. [Online]. *2007(06/21),* Available: http://www.uddi.org/faqs.html

[9] HTTP - hypertext transfer protocol. [Online]. *2007(06/21),* Available: http://www.w3.org/Protocols/

[10] D. J. Bernstein. SMTP: Simple mail transfer protocol. [Online]. *2007(06/21),* Available: http://cr.yp.to/smtp.html

[11] R. H. Katz, "Adaptation and mobility in wireless information systems," *IEEE Personal Communications,* vol. 1, pp. 6-17, 1994.

[12] M. Satyanarayanan, "Fundamental challenges in mobile computing," in *Proceedings of the 1996 15th Annual ACM Symposium on Principles of Distributed Computing,* Philadelphia, PA, USA, 1996, pp. 1-7.

[13] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi and S. Weerawarana, "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI," *IEEE Internet Comput.,* vol. 6, pp. 86-93, 03. 2002.

[14] Business process execution language for web services version 1.1. [Online]. *2007(06/21),* pp. 136. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf

[15] M. P. Papazoglou, P. Traverso, S. Dustdar and F. Leymann. Service-oriented computing research roadmap. *2007(06/21),* pp. 29. Available: ftp://ftp.cordis.europa.eu/pub/ist/docs/directorate_d/st-ds/services-research-roadmap_en.pdf

[16] OWL-S: Semantic markup for web services. [Online]. *2007(06/21),* pp. 25. Available: http://www.daml.org/services/owl-s/1.0/owl-s.pdf

[17] L. B. Mummert, M. R. Ebling and M. Satyanarayanan, "Exploiting weak connectivity for mobile file access," in *Fifteenth ACM Symposium on Operating Systems Principles,* Copper Mountain, Colorado, 1995, pp. 143-155.

[18] L. Kleinrock, "Nomadic computing," in *Third INFORMS Telecommunications Conference,* Boca Raton, Florida, 1997, pp. 5-15.

[19] D. Barbara and T. Imielinski, "Sleepers and workaholics: Caching strategies in mobile environments," in *1994 ACM SIGMOD International Conference on Management of Data,* Minneapolis, Minnesota, 1994, pp. 1-12.

[20] G. V. Chockler, D. Dolev, R. Fredman and R. Vitenberg, "Implementing a caching service for distributed CORBA objects," in *Proceedings of Middleware 2000,* Huston River Valley, New York, 2000, pp. 1-23.

[21] N. Kouici, D. Conan and G. Bernard, "Caching components for disconnection management in mobile environments," *Lecture Notes in Computer Science,* pp. 1322-1339, 25-29 Oct. 2004. 2004.

[22] R. Friedman, "Caching web services in mobile ad-hoc networks: Opportunities and challenges," in *Proceedings of the Second International Workshop on Principles of Mobile Commerce POMC '2002,* Toulouse, France, 2002, pp. 90-96.

[23] OASIS. Reference model for service oriented architecture 1.0. [Online]. *2007(06/21),* pp. 31. Available: http://www.oasis-open.org/committees/download.php/18486/pr-2changes.pdf

[24] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," in *Proceedings of International Conference on Software Engineering,* Limerick, Ireland, 2000, pp. 407-416.

[25] D. Terry and V. Ramasubramanian, "Caching XML Web services for mobility," ACM *Queue,* vol. 1, pp. 70-78, 05. 2003.

[26] K. Elbashir and R. Deters, "Nomadic web service clients," in *9th International Database Engineering and Applications Symposium (IDEAS 2005),* Montreal, Canada, *2005*, pp. 379-388.

[27] J. Wang, "A survey of Web caching schemes for the Internet," Computer *Communication Review,* vol. 29, pp. 36-46, 10. 1999.

[28] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel and D. C. Steere, "Coda: a highly available file system for a distributed workstation environment," *IEEE Trans. Comput.,* vol. 39, pp. 447-459, 04. 1990.

[29] R. Bakalova, A. Chow, C. Fricano, P. Jain, N. Kodali, D. Poirier, S. Sankaran and D. Shupp, "WebSphere Dynamic Cache: Improving J2EE application performance," *IBM Syst J,* vol. 43, pp. 351-370, 2004.

[30] H. Chang, C. Tait, N. Cohen, M. Shapiro, S. Mastrianni, R. Floyd, B. Housel and D. Lindquist, "Web browsing in a wireless environment: Disconnected and asynchronous operation in ARTour web express," in *Proceedings of Third ACM/IEEE International Conference on Mobile Computing and Networking 1997 (MobiCom'97),* Budapest, Hungary, 1997, pp. 260-269.

[31] S. Dar, M. J. Franklin, B. T. Jónsson, D. Srivastava and M. Tan, "Semantic data caching and replacement," in *The 22th International Conference on VLDB,* Bombay, India, 1996, pp. 330-341.

[32] Q. Ren, M. H. Dunham and V. Kumar, "Semantic caching and query processing," *IEEE Trans. Knowled. Data Eng.,* vol. 15, pp. 192-210, 2003.

[33] Q. Ren and M. H. Dunham, "Using semantic caching to manage location dependent data in mobile computing," in *Sixth Annual International Conference on Mobile Computing and Networking,* Boston, Massachusetts, USA, 2000, pp. 210-221.

[34] B. Zheng and Dik Lun Lee, "Semantic caching in location-dependent query processing," *SSTD,* pp. 97-113, 12-15 July 2001.

[35] J. Yin, L. Alvisi, M. Dahlin and A. Iyengar, "Engineering server-driven consistency for large scale dynamic web services," in *WWW '01: Proceedings of the 10th International Conference on World Wide Web,* Hong Kong, China, 2001, pp. 45-57.

[36] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems,* Monterey, California, USA, 1997, pp. 193-206.

[37] J. Griffioen and R. Appleton, "Improving file system performance via predictive caching," in *Parallel and Distributed Computing Systems,* Vancouver, B.C.1995, pp. 165-170.

[38] J. Griffioen and R. Appleton, "Reducing file system latency using a predictive approach," in *Summer USENIX Technical Conference,* Boston, Massachusetts, USA, 1994, pp. 197-207.

[39] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve World Wide Web latency," *Computer Communication Review,* vol. 26, pp. 22-36, 1996.

[40] Z. Jiang and L. Kleinrock, "Web prefetching in a mobile environment," *IEEE Personal Communications,* vol. 5, pp. 25-34, 10. 1998.

[41] Z. Jiang and L. Kleinrock, "Prefetching links on the WWW," in *Proceedings of ICC'97 - International Conference on Communications,* Montreal, Quebec, Canada, 1997, pp. 483-489.

[42] Workflow Management Coalition. (1999, The workflow management coalition terminology & glossary. [Online]. *2007(06/25),* pp. 65. Available: http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf

[43] J. L. Zhao and A. Kumar, "Data management issues for large scale, distributed workflow systems on the Internet," *Data Base for Advances in Information Systems,* vol. 29, pp. 22-32, 1998.

[44] J. Jing, K. Huff, H. Sinha, B. Hurwitz and B. Robinson, "Workflow and application adaptations in mobile environments," in *Proceedings of WMCSA99: 2nd IEEE Workshop on Mobile Computing Systems and Applications,* 1999, pp. 62-69.

[45] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky and J. Zelenka, "Informed prefetching and caching," in *Fifteenth ACM Symposium on Operating Systems Principles,* New Orleans, Louisiana, USA, 1995, pp. 79-95.

[46] R. H. Patterson, G. A. Gibson and M. Satyanarayanan, "Using transparent informed prefetching (TIP) to reduce file read latency," in *Proceedings of the Goddard Conference on Mass Storage Systems and Technologies,* Greenbelt, MD, USA, 1992, pp. 329-342.

[47] G. Sharifi, J. Vassileva and R. Deters, "Seamless communication and access to information for mobile users in a wireless environment," in *ICEIS 2004 - Proceedings of the Sixth International Conference on Enterprise Information Systems*, Porto, Portugal, 2004, pp. 122-129.

[48] T. Takase and M. Tatsubori, "Efficient web services response caching by selecting optimal data representation," in *The 2nd IEEE International Conference on Distributed Computing Systems,* Tokyo, Japan, 2004, pp. 188-197.

[49] K. Devaram and D. Andresen, "SOAP optimization via client side caching," in *Proceedings of the International Conference on Web Services, ICWS'03,* Las Vegas, Nevada, USA, 2003, pp. 520-524.

[50] S. Seltzsam, R. Holzhauser and A. Kemper, "Semantic caching for web services," in *Proceedings of the Third International Conference.Service-Oriented Computing - ICSOC 2005,* Vienna, Austria, 2005, pp. 324-340.

[51] V. Ramasubramanian and D. Terry. (December 2004, Caching of XML web services for disconnected operation. [Online]. Available: http://www.cs.cornell.edu/People/ramasv/WebServiceCache/WebServiceCache(techrepor t).pdf

[52] DAML+OIL. [Online]. *2007(06/21),* Available: http://www.daml.org/2000/12/daml+oil-index.html

[53] OWL web ontology language overview. [Online]. *2007(06/21),* Available: http://www.w3.org/TR/owl-features/

[54] D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness and P. F. Patel-Schneider, "OIL: an ontology infrastructure for the Semantic Web," *IEEE Intelligent Systems,* vol. 16, pp. 38-45, 03. 2001.

[55] Resource description framework (RDF). [Online]. *2007(06/21),* Available: http://www.w3.org/RDF/

[56] RDF vocabulary description language 1.0: RDF schema. [Online]. *2007(06/21),* Available: http://www.w3.org/TR/rdf-schema/

[57] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann and I. Horrocks, "The Semantic Web: the roles of XML and RDF," *IEEE Internet Comput.,* vol. 4, pp. 63-73, 2000.

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

[59] G. Alonso, R. Gunthor, M. Kamath, D. Agrawal, A. El Abbadi and C. Mohan, "Exotica/FMDC: a workflow management system for mobile and disconnected clients," Distributed *and Parallel Databases,* vol. 4, pp. 229-247, 07. 1996.

[60] P. Wohed, van der Aalst, W.M.P., M. Dumas and A. H. M. ter Hofstede, "Analysis of web services composition languages: The case of BPEL4WS," in *Proceedings,* 2003, pp. 200-15.

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

[62] T. Karagiannis, M. Molle, M. Faloutsos and A. Broido, "A nonstationary poisson view of internet traffic," in *IEEE INFOCOM 2004 - Conference on Computer Communications - Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies,* Hong Kong, China, 2004, pp. 1558-1569.

[63] Poisson distribution. [Online]. *2007(07/16),* Available: http://en.wikipedia.org/wiki/Poisson_distribution

[64] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," Proceedings - *IEEE INFOCOM,* vol. 1, pp. 126-134, New York, NY, USA, 1999.

[65] .NET compact framework. *2007(06/25),* Available: http://msdn2.microsoft.com/es-ar/netframework/aa497273.aspx

[66] JAVA. [Online]. *2007(06/21),* Available: http://java.sun.com/

[67] Apache tomcat. [Online]. *2007(06/21),* Available: http://tomcat.apache.org/

[68] Web services - axis. [Online]. *2007(06/21),* Available: http://ws.apache.org/axis/

[69] MySQL. [Online]. *2007(06/24),* Available: http://www.mysql.com/