OUKA
The University of Osaka
Institutional Knowledge Archive

Title	Design and Optimization of Protocols for Distributed Data Processing Services on Overlay and Mobile Networks
Author(s)	Sakai, Yuki
Citation	大阪大学, 2013, 博士論文
Version Type	VoR
URL	https://hdl.handle.net/11094/27481
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

情学 16079

Design and Optimization of Protocols for Distributed Data Processing Services on Overlay and Mobile Networks

January 2013

Yuki SAKAI

Design and Optimization of Protocols for Distributed Data Processing Services on Overlay and Mobile Networks

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2013

Yuki SAKAI

List of Major Publications

Journal Papers

1. Yuki Sakai, Akihito Hiromori, Hirozumi Yamaguchi and Teruo Higashino, "Design and Development of Service Execution Platform for Overlay Networks", *IPSJ Journal*, Vol.53, No.11, pp.2612–2623, November 2012 (in Japanese).

2. Yuki Sakai, Akira Uchiyama, Hirozumi Yamaguchi and Teruo Higashino: "Self-Estimation of Neighborhood Distribution for Mobile Wireless Nodes", *IPSJ Journal*, Vol.54, No.2, April 2013.

Conference Papers

1. Yuki Sakai, Akihito Hiromori, Hirozumi Yamaguchi, Khaled El-Fakih and Teruo Higashino : "An Integrated Tool for Development of Overlay Services", *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (SIMUTools ' 09)*, Article No.61, Roma, Italy, March 2009.

2. Yuki Sakai, Akihito Hiromori, Hirozumi Yamaguchi and Teruo Higashino: "A Study on Designing Overlay for Ubiquitous Services", *Proceedings of the 5th International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2010)*, Seattle, USA, April 2010.

Abstract

The popularity of well-provisioned sensors have enhanced to research and develop the systems for new kinds of services called ubiquitous services to make our life safe, secure, and comfortable by collecting, aggregating and processing sensing data distributed to urban areas. Furthermore, recent innovation of wireless communication technology has brought us possibilities to deploy infrastructure-less wireless applications. The spread of modern mobile sensing devices such as smartphones and on-board units for vehicles, and the advance of the technology should accelerate to innovate a lot of situation-aware services. When we develop such systems that provide these services, the following issue might stand in our way, how to collect and process a flood of context data from mobile sensors. Since the amount of sensing data is quite enormous, data processing should be executed on multiple servers and devices collaboratively to reduce their workloads of servers and devices. On the other hand, sensing data need to be collected essentially since sensors are geographically distributed to the urban areas over the wireless sensor networks. However, due to constraints from the network bandwidth or computational power limitation, workloads of the network should be reduced as much as possible. To provide services cost-efficiently, we must overcome this issue comprehensively.

Considering the above issue, this thesis studies the following two study topics: (1) a service design technique reducing both of the network and server loads on an overlay network which is composed of repositories storing the sensing data and servers to process their data, and (2) a pure-serverless service protocol which can be executed on a mobile ad-hoc network, and therefore in which there is no workload of servers essentially.

As the first topic of this thesis, we propose a method that derives an optimal service execution on overlay networks, satisfying constraints from networks and servers. For example, we consider a traffic forecasting service using movie data from fixed cameras on the streets in urban areas and GPS

data from vehicles there. In this service, we assume that the movie data and the GPS data should be stored at a local server in each location. The service analyzes data to measure the trip time of vehicles and estimates their trajectories, and finally obtains the amount of traffic in the entire area. This service transfers data among servers and processes data alternately. Generally, we need to address the following two challenges to design the service. To avoid the bottlenecks of processing time, we should design the service considering capabilities of servers, network capacity and the amount of data to transfer. However, it is hard for service designers to obtain the optimal design that reduces workloads of networks and servers. (ii) Additionally, servers and network loads may change over time according to their utilization. However, maintaining the reasonable performance according to the dynamic changes of traffic volume and system loads is not an easy task.

To tackle the challenge (i), we propose a method to derive execution sequences of a given application-level service that is executed by cooperative servers on overlay networks. We assume that a given service is composed of service components which indicate service processing units. We also assume that the service is modeled as a coloured Petri net [1] which can be described as the combination of sequential tasks, parallel tasks, branch operations and synchronization operations. It can derive an optimal allocation of components to achieve minimum response time avoiding overloading network and servers.

As for the challenge (ii), we design and develop a service execution platform for supporting the development of overlay services. Using the platform, services can be installed and executed automatically on real networks when the service descriptions and network information are given.

We have conducted experiments on PlanetLab to validate our method. As an application example, we have developed three applications using the proposed platform: a video transcoding service, a traffic forecasting and a probe-vehicle data analysis. The experimental results have shown that the proposed method could derive efficient execution sequences and they could achieve higher throughput than the other methods. Finally, we confirm that we can easily conduct experiments on real network by the proposed platform and we can instantly monitor the loads of the network and servers.

In the second study, we focus on collaborative tasks processing in modern networks with many small, mobile clients. This approach aims at reducing workloads of a network by avoiding transferring data to servers. As an

example of a service in this situation, we investigate an estimation protocol to obtain the distribution of mobile nodes in their surroundings.

Ad-hoc wireless communication is a cost-efficient way of data fusion and diffusion among local agents. In particular, if pedestrians can estimate and obtain the information on their surroundings in real-time through ad-hoc communication, many services and applications can be provisioned without infrastructures. For example, it would be more beneficial to a human navigation system for emergency evacuation and to stranded commuters in disasters if information on the distribution of people in their surroundings is available. Some literature have proposed methods for people density estimation in urban areas. For example, in Ref. [2], Bluetooth scan is used for estimating the number of nearby nodes. Ref. [3] has investigated people density estimation using locations of mobile phones obtained via base stations for large scale urban monitoring. However, real-time estimation of mobile node distribution by collaboration through ad-hoc networks is still challenging.

We propose a method for mobile wireless nodes, which are pedestrians, to estimate the distribution of mobile nodes in their surroundings. In the proposed method, each node is assumed to know its location roughly (*i.e.* within some error range) and to maintain a density map covering its surroundings. This map is updated when a node receives a density map from a neighboring node. Each node also updates its density map in a timely fashion by estimating the change of node distribution over time due to node mobility. The goal of our study is to propose an autonomous protocol to let mobile nodes have accurate node distribution with reasonable amount of wireless ad-hoc communication traffic.

The simulation experiments have been conducted and the similarity between the real and estimated distributions has been measured. The results in three different scenarios have shown that the proposed method could attain average localization errors less than 10m.

In this thesis, we propose two protocols for distributed data processing services to reduce workloads of a network and devices as much as possible. As the first protocol, we propose a method that derives an optimal service execution on overlay networks, satisfying constraints from networks and servers. In the second protocol, we focus on collaborative tasks processing in modern ad-hoc networks with many small, mobile clients.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The popularity of well-provisioned sensors have enhanced to research and develop the systems for new kinds of services called ubiquitous services to make our life safe, secure, and comfortable by collecting, aggregating and processing sensing data distributed to urban areas. Furthermore, recent innovation of wireless communication technology has brought us possibilities to deploy infrastructure-less wireless applications. The spread of modern mobile sensing devices such as smartphones and on-board units for vehicles, and the advance of the technology should accelerate to innovate a lot of situation-aware services. When we develop such systems that provide these services, the following issue might stand in our way, how to collect and process a flood of context data from mobile sensors. Since the amount of sensing data is quite enormous, data processing should be executed on multiple servers and devices collaboratively to reduce their workloads. On the other hand, sensing data need to be collected essentially since sensors are geographically distributed to the urban areas over the wireless sensor networks. However, due to constraints from the network bandwidth or computational power limitation, workloads of the network should be reduced as much as possible. To provide services cost-efficiently, we must overcome this issue comprehensively.

Considering the above issue, this thesis studies the following two study topics: (1) a service design technique reducing both of the network and server loads on an overlay network which is composed of repositories storing the sensing data and servers to process these data, and (2) a pure-serverless

service protocol which can be executed on a mobile ad-hoc network, and therefore in which there is no workload of servers essentially.

As the first topic of this thesis, we propose a method that derives an optimal service execution on overlay networks, satisfying constraints from networks and servers. For example, we consider a traffic forecasting service using movie data from fixed cameras on the streets in urban areas and GPS data from vehicles there. In this service, we assume that the movie data and the GPS data should be stored at a local server in each location. The service analyzes data to measure the trip time of vehicles and estimates their trajectories, and finally obtains the amount of traffic in the entire area. This service transfers data among servers and processes data alternately. Generally, we need to address the following two challenges to design the service. (i) To avoid the bottlenecks of processing time, we should design the service considering capabilities of servers, network capacity and the amount of data to transfer. However, it is hard for service designers to obtain the optimal design that reduces workloads of networks and servers. (ii) Additionally, servers and network loads may change over time according to their utilization. However, maintaining the reasonable performance according to the dynamic changes of traffic volume and system loads is not an easy task.

To tackle the challenge (i), we propose a method to derive execution sequences of a given application-level service that is executed by cooperative servers on overlay networks detailed in Chapter 3. We assume that a given service is composed of service components which indicate service processing units. This method can determine an optimal protocol that each service component should be executed. We also assume that the service is modeled as a coloured Petri net [1] which can be described as the combination of sequential tasks, parallel tasks, branch operations and synchronization operations. In particular, we use CPN Tools [4] for writing services in a centralized and network-independent way. Then, our method analyzes a given service description and automatically derives a distributed version of the service taking into consideration the targeted overlay network specification. It can derive an optimal allocation of components to achieve minimum response time avoiding overloading network and servers.

As for the challenge (ii), we design and develop a service execution platform for supporting the development of overlay services detailed in Chapter 4. Using the platform, services can be installed and executed automatically on real networks when the service descriptions and network information are given. The platform can also measure network loads and CPU loads continuously, and can adaptively determine new allocation optimizing the system load in the environment. On the platform, the distributed version of the service is interpreted and the overlay nodes are asked to execute the service as specified in this description. During that time, the utilization of overlay links and occupation of processors is monitored so that related information can be provided to the developers. In particular, the proposed platform is equipped with the resource optimizer, which determines the best way to execute the service in the targeting overlay network to maximize the performance of the service. As a result, service designers can easily manage the service execution.

Using the proposed platform, we have conducted experiments on PlanetLab to validate our method. As an application example, we have developed three applications using the proposed platform: a video transcoding service, a traffic forecasting and a probe-vehicle data analysis. The experimental results have shown that the proposed method could derive efficient execution sequences which could achieve higher throughput than the other sequences. Finally, we confirm that we can easily conduct experiments on real network by the proposed platform and we can instantly monitor the loads of the network and servers.

Several frameworks and related toolsets have been proposed and developed for similar or different objectives. For designing service overlay, Refs. [5, 6, 7] have dealt with service design methodologies, and Refs. [8, 9, 10, 11, 12] have developed some support tools for development of overlay networks. Compared with these methodologies or tools, our contribution is summarized as follows. First, we provide a toolset with several unique features in supporting high level design of service overlay. Using the algorithm presented in Ref.[13], the tool can automatically derive a distributed service description from a given centralized service description written in high-level

Petri nets. Also the tool has the resource optimizer that allows to optimize the performance of the service. By these features, developers can design the optimized services using powerful GUI from CPN Tools, without knowing the overlay network specification. Second, the tool has several functions for program deployment, debug and performance monitoring. Finally, using a realistic example service, we have conducted experiments in real distributed environment to demonstrate the advantages of our tool. A more detailed discussion on the advantages of our integrated environment along with some related methodologies will be given in Section 2.1.

In the second study, we focus on collaborative tasks processing in modern networks with many small, mobile clients. This approach aims at reducing workloads of a network by avoiding transferring data to servers. As an example of a service in this situation, we investigate an estimation protocol to obtain the distribution of mobile nodes in their surroundings. In Intelligent Transportation Systems (ITS), many research efforts have been conducted for situation awareness of pedestrians and vehicles based on DSRC for collision avoidance. For example, OKI has developed a DSRC attachment for mobile phones for pedestrian safety[14] by broadcasting positions of pedestrians obtained by GPS. People centric sensing[15] is also an emerging technology using sensing information such as traffic information from smartphones for urban sensing.

These studies indicate that ad-hoc wireless communication is a cost-efficient way of data fusion and diffusion among local agents. In particular, if pedestrians can estimate and obtain the information on their surroundings in real-time through ad-hoc communication, many services and applications can be provisioned without infrastructures. For example, it would be more beneficial to a human navigation system for emergency evacuation and to stranded commuters in disasters if information on the distribution of people in their surroundings is available. Some literature have proposed methods for people density estimation in urban areas. For example, in Ref. [2], Bluetooth scan is used for estimating the number of nearby nodes. Ref. [3] has investigated people density estimation using locations of mobile phones obtained via base stations for large scale urban monitoring. In spite of these

researches, real-time estimation of mobile node distribution by collaboration through ad-hoc networks is still challenging. To aggregate the information on their surroundings to servers may be cost-efficient way to provide density estimation. However, this way can cause a scalability concern and cannot provide the estimation in provisional situation such as festivals, disasters and so on.

In Chapter 5, we propose a method for mobile wireless nodes, which are pedestrians, to estimate the distribution of mobile nodes in their surroundings. In the proposed method, each node is assumed to know its location roughly (*i.e.* within some error range) and to maintain a density map covering its surroundings. This map is updated when a node receives a density map from a neighboring node. Each node also updates its density map in a timely fashion by estimating the change of node distribution over time due to node mobility. The goal of our study is to propose an autonomous protocol to let mobile nodes have accurate node distribution with reasonable amount of wireless ad-hoc communication traffic.

For estimating the node distribution, a mobile node may independently maintain and share positions of each node. However, the amount of data exchanged among mobile nodes may be large since a large number of nodes are expected in urban areas. To keep the size of the data constant, we use a density map where we divide a target region into square cells and the expected number of nodes (i.e. density) in each cell is maintained. Nodes can estimate the current distribution of nodes from their own density maps by finding cells with a high density in a greedy fashion. To build a density map, with a certain interval, each node broadcasts its own density map where its *area of presence* (the area in which a true location is included) is merged. On receiving a density map from neighboring nodes, the node updates such a part of its own density map by the received density information which seems to be fresher.

We note that there is a clear trade-off between the freshness of density information and the required amount of wireless capacity to exchange density information. To pursue this trade-off, we have two key ideas. First, we provide an *estimation function* that estimates the future density map

based on its time-varying characteristics. As a simple example, if we know the maximum speed V_{max} of mobile nodes, an estimation function that estimates the density map after Δt time can be designed in such a way that each density in the current map is spread over $V_{max} \cdot \Delta t$ region. Another function can be designed in such a way that the density is spread only to the directions toward which other nodes exist. This is based on the property that pedestrians walk on roads. Second, we design an adaptive protocol that controls the transmission intervals of messages depending on the density of surroundings, in order to avoid similar density maps to be emitted to the wireless channel.

The simulation experiments have been conducted and the similarity between the real and estimated distributions has been measured. The results in three different scenarios have shown that the proposed method could attain average localization errors less than 10m.

The rest of this thesis is organized as follows. The next chapter reviews the related work. Chapter 3 details our method deriving optimal service execution on overlay networks. Chapter 4 describes a design of the service execution platform for overlay networks. Chapter 5 details estimation protocol of distribution for mobile wireless nodes. Finally, Chapter 6 summarizes this thesis.

Chapter 2

Related Work

2.1 Service Models and Service Designing Frameworks

As service models to describe collaboration with components, a execution path [5] which simply describes sequence tasks and a directed acyclic graph(DAG) [6, 7] are proposed. Compared with a execution path and the directed acyclic graph, a coloured Petri net can treat more realistic services. A coloured Petri net can describe more complicate controls such as loop operations and branch operations, or more specifically, it can describe synchronization operations such as references of databases and change processing tasks of the service depending on requests.

Design techniques for distributed systems supported by computers are proposed in Ref.[8, 9, 10, 11, 12, 16, 17]. iOverlay[8] provides a distributed platform on an overlay network which can develop and evaluate overlay services. This platform introduces a message communication framework among servers to support service design. Arigatoni[9] provides a distributed service design framework which discover resources enough to execute services automatically. Ref.[10, 11] focus on reducing designing cost for development of distributed services. MACEDON [12] provides both of simulator and a testbed such as PlanetLab so that MACEDON can support to analyze and evaluate distributed services on a P2P network. Ref.[18] proposes a programing language extended Guarded Horn Clauses for a distributed environment on an overlay network. On the other hand, evaluation methods

for service performance on an overlay network using a network simulator are proposed. For example, OverSim [19] is proposed as a network simulator specialized in an overlay network. Platforms supporting design and development of a service by network simulations [20, 21] are also proposed. Specifically, a platform proposed in Ref.[20] can develop not only traditional API-level services but also high-level services such as behavior level services.

In cloud computing, many resource optimization methods on cloud networks [22, 23, 24, 25] are proposed. In cloud computing for mobile wireless network, Ref.[26] is presented a middleware of mobile cloud services for Android devices. The high-level architecture and functional requirements for a mobile sensing service are presented in Ref.[27]. Marinelli [28] proposes a platform which supports cloud computing on Android devices. Ref.[29] predicts the future environment in 2022 such as device capabilities and networks and proposes mClouds architecture which supports mobile cloud computing. Sonora [30] focuses on fault tolerance and presents a platform supporting dynamic adaptation and failure recovery.

Compared with the above methodologies, in our work, we deal with services that are decoupled from overlay network specifications. A service can be written in a centralized way, using a known Petri net model, and developers can start building the design without being aware of overlay network architecture. The distributed version of the given service description is automatically derived to alleviate developers' load in writing distributed programs directly. Also, the resource optimizer allows optimizing the performance of the service using some given objective functions. Several objective functions are provided for the convenience of deployment of programs with better performance. A toolset that integrates my work with existing methods and tools is provided.

2.2 Data Aggregation Techniques for Mobile Wireless Nodes

In Vehicular Ad-hoc NETworks (VANETs), there have been various approaches to aggregate and disseminate several types of contexts like road surface condition, temperature, traffic jam information [31, 32, 33, 34]. Sim-

ilar approaches have been considered in Wireless Sensor Networks (WSNs) [35, 36, 37, 38]. Some of them consider reducing the amount of data based on its similarity (*i.e.* elimination of data redundancy) and others consider in-network computing of given queries.

Our proposed method falls into these categories in the sense that it is aimed at aggregating (sensed) data with less amount of traffic. However, the proposed method is designed for mobile nodes to self-estimate their neighborhood distribution. Therefore, the data is time-varying in the scale of minutes while VANETs and WSNs target aggregation of data such as load surface condition and wide-area traffic condition information which are relatively stable in long-term. Hence, we have to consider the trade-off between timeliness of data of mobile nodes' locations and traffic overhead. We note that object detection and tracking in WSNs have to deal with real-time motion of objects (thus the data must be time-varying in very short term). However, these applications are not aimed at aggregating data but detecting objects.

As we stated in the introduction, each node has estimation functions to estimate the dynamic change of node distribution, and exchanges the estimated result with others to help increase the accuracy of density maps. Also depending on the neighborhood density, each node controls the transmission interval. Based on these two ideas, we have designed a protocol that deals with a unique problem, that is, self-estimation of mobile node distribution. In this sense, our approach is original.

Our goal relates to localization algorithms [39, 40, 41, 42], which aim to estimate positions of nodes. However, the goal of localization algorithms is to estimate each node's position *by itself* and does not much care about positions of other nodes. Also their main concern is accuracy, while our challenge is to design a protocol that pursues the trade-off between the accuracy and traffic.

There are several methods for estimating density of people in urban areas[43, 44, 45]. Mobile space statistics [45] presented by NTT DOCOMO tracks populations of each area by counting mobile phone users observed at each base station. However, this approach aims at large-scale statistics such

as the population in a city, which is different from our target. Ref. [43] proposes a method to reconstruct people flow from existing person-trip survey data. Ref. [44] proposes a method for density estimation using coarse location information obtained from mobile phone call data. To the best of our knowledge, there is no research to provide real-time estimation of node density in urban areas using cooperation among mobile users. A straightforward approach is to upload position data obtained by GPS from all the nodes by using 3G networks. However, the 3G network traffic is overloaded in such an approach particularly in urban areas where a large number of people exist. In contrast, we use ad hoc communication between mobile nodes to share density information while avoiding 3G network overloading.

Chapter 3

A Method Deriving Optimal Service Execution on Overlay Networks

3.1 Introduction

In this chapter, we propose a method to derive execution sequences of a given application-level service that is executed by cooperative servers on overlay networks. The proposed method assumes that a service consists of service components, and it can derive optimal allocation of components that does not overload network links and servers. Using the platform, services can be installed and executed easily on real networks. We have conducted experiments on PlanetLab to validate our method. The experimental results have shown that the proposed method could derive efficient execution sequences and they could achieve higher throughput than the other sequences.

The rest of this chapter is organized as follows. Section 3.2 presents definition and description of a service, and Section 3.4 presents definition of an overlay network. Section 3.5 describes service execution on an overlay network and optimization algorithm. Section 4.2 shows the design and implementation of the proposed platform and Section 3.6 shows experimental results. Finally, this chapter is concluded in Section 3.7.

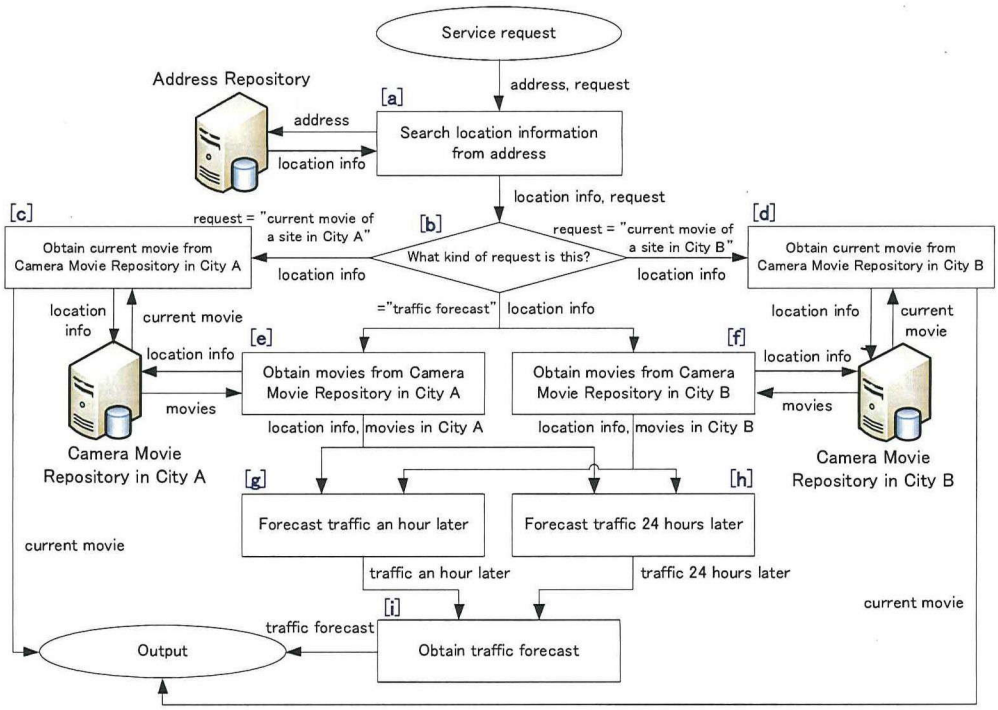


Figure 3.1: Service Example (transportation analysis services).

3.2 Definition of Services and an Example of Description

In this chapter, we assume that the service is a transaction composed of data processing units called service components which have sensing data as inputs and it can be described as the combination of sequence tasks, parallel tasks, branch operations and synchronization operations among service components.

Fig.3.1 is an example of a service. This example is a flowchart-like description of a traffic analysis service using movie data from fixed cameras at the streets in two cities: City A and City B. Given a request which have a target address as one of the arguments, the service provides trraffic forecasts around the address in one hour and 24 hours by the following process. At first, the component [a] in Fig.3.1 searches the database(called a repository

hereafter) which manages the locations of cameras for the information of cameras around the target address. Next, using the searching result, component [e] and component[f] search repositories for movie data which store the camera movies in the City A and City B respectively. Then, analyzing movie data, component [g] and component [h] forecast the traffic at the address in one hour and in 24 hours respectively. Finally, this service send back the forecasting results to the user of the request. Additionally, the component [i] can let the service perform different processes from the traffic forecasting depending on a request. If an user want to know the current traffic of his target address, component [c] and component [d] search the repositories, return a current movie to him. Component [c] will be executed when the target address is in City A. Component [d] wiil be executed when it is in City B as well.

In our method, we assume that a request in this service arises transaction processing mentioned above and the service need to respond to the flood tide of requests from many users simultaneously. We define that the service is actually described as a coloured Petri net like Fig.3.2. The coloured Petri nets are composed of transitions which represent processing data, places which represent buffers or storages of data and coloured tokens which represent data. A transition needs sufficient coloured tokens at input places to execute the transition. On the execution, the coloured tokens at each input place and new coloured tokens are generated at each output place. A transition can have a conditional expression depending on the value of coloured tokens to control the execution.

Coloured Petri nets is described in 3.3 in details. For more detail, please refer Ref.[1].

In this chapter, we assume that requests of a service are represented by coloured tokens, service components ([a] thru [i] in Fig.3.1) are represented by transitions($T1$ thru $T9$ in Fig.3.2), and memory areas of data used by transitions are represented by places($P1$ thru $P11$). For example, as to transition $T1$ (component [a]), place $P1$ is an input place and place $P2$ is an output place. A coloured token described as $\langle "squareatStationA", "trafficforecast" \rangle$ is given as a request. On executing $T1$, this token is deleted. Next, repos-

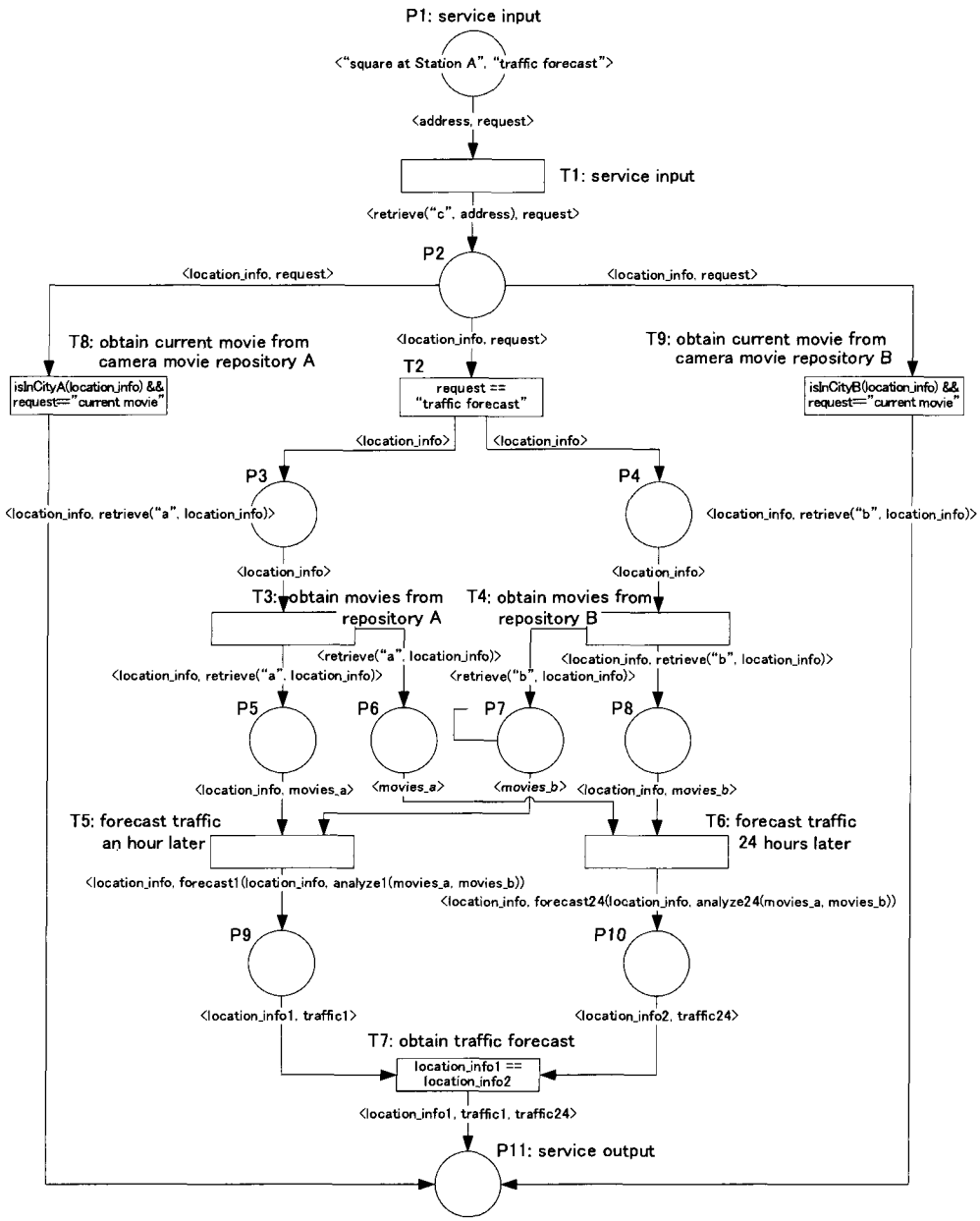


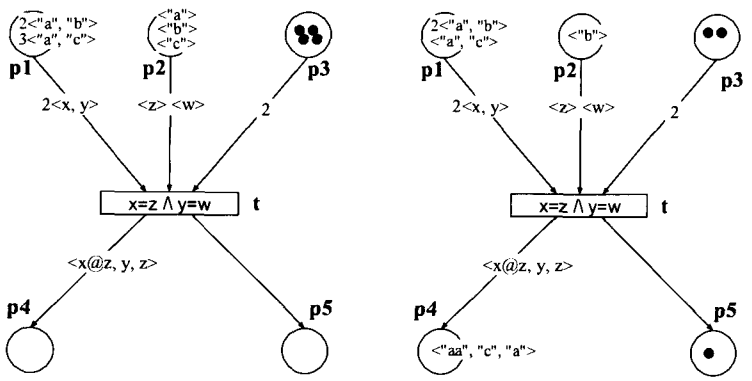
Figure 3.2: Transportation Analysis Services of Fig.3.1 Formally Written in Coloured Petri Net.

itory "C" (place $P12$) is searched for the location information of cameras around the target address based on a label $\langle retrieve("c", address), request\rangle$

put on output edge $(T1, P2)$ called an arc. Then, a new coloured token including the searching result is generated on place $P2$. Place $P1$, place $P12$, place $P13$ and place $P14$ represent location info repository "C", camera movie repository "A" and camera movie repository "B" respectively and the tokens of these repositories indicate data of the repositories. For instance, data of repository "C" is described as coloured token $\langle C \rangle$ held by place $P12$. Place $P2$ holds searching results of location information, place $P5$ and place $P8$ hold searching results of camera movies, and place $P9$ and place $P10$ hold results of traffic forecasting. Place $P3$ and place $P4$ represent buffers to wait for searching process of the camera movies. The results of a series of this transaction processing are held by place $P11$.

If a movement of data through the execution of components connected in order of a sequence is regarded as a data flow, a transition which hold multiple input places like transition $T5$ and transition $T6$ indicates a synchronization of a data flow since the transition is executable only when all of the input places of the transition hold a coloed token. If a transition which hold multiple output places like transition $T2$, the execution of the transition indicates start of parallel processing as well. A place holding multiple output transitions indicates a branch and which transition to be executed can be controlled depending on the value of coloured tokens. For example, the conditional expression of transition $T2$ become true and transition $T2$ can be executed when place $P2$ holds a coloured token which value of the field *request* is a string of "traffic forecast". On the other hand, transition $T8$ is executable when *request* is "current movie" and the target address is in City A, and transition $T9$ is executable when *request* is "current movie" and the target address is in City B.

In this chapter, in terms of structured constraints of the service model, we assume that a reference of a repository is only described as a self-loop which connects the transition representing the reference process and the place representing the repository mutually. A self-loop is defined that the input place and the output place of a transition is the same. For instance, transition $T1$ have a self-loop with place $P12$ (repository "C") to refers the repository. On execution of transition $T1$, coloured token $\langle C \rangle$ is given as



(a) Before firing of t. (b) After firing of t.

Figure 3.3: An Example of Coloured Petri Net.

an input of the transition. After the execution, the process of the reference to repository "C" is finished when coloured token $\langle C \rangle$ is returned to the place. It is also assumed that no service includes a directed cycle except a self-loop to describe a repository due to the optimization algorithm mentioned in Section 3.5. Additionally, we assume that every service includes place p_{in} to put requests from service users initially and place p_{out} to put results of requests. In Fig.3.2, place $P1$ corresponds to place p_{in}, place $P11$ corresponds to place p_{out} respectively.

3.3 Coloured Petri Net

In this chapter, we deal with services written as coloured Petri nets. Coloured Petri nets are extended Petri nets where tokens have values and the firability of transitions may depend on those values. I note that Coloured Petri Net (CPN) [1] is a known high-level Petri net that falls into this category. These models have enough modeling power, analytical power and tool support (such as CPN Tools [4]) to specify, verify and analyze large and practical software systems [46], communication protocols [47, 48], control systems and so on [1, 49].

In Petri nets, a place (denoted as a circle) and a transition (denoted as a rectangle) may represent data (or system state) and a task, respectively.

A place and a transition may be connected by a directed edge called an arc (denoted by an arrow). Tokens (denoted as black dots) in places represent the current state of the system, and execution ("firing" in the Petri net terminology) of a transition may consume/produce tokens from/to the places connected to the transition.

Formally, in coloured Petri nets, each incoming arc to transition t from place p has a label (called an *arc label*) of the form of $k_1X_1k_2X_2...$ where k_i is a positive integer, X_i is a n-tuple of variables like $\langle x_1, x_2, ...x_n \rangle$ and n is an arbitrary non-negative integer assigned to place p. Place p may have tokens, each of which is a n-tuple of values $C_i = \langle c_1, c_2, ...c_n \rangle$. A set of tokens which can be assigned to the label of an incoming arc to transition t is called an *assignable set* of the arc. Moreover, a transition t may be associated with a logical formula of variables from the labels of incoming arcs of t, called a *condition*. Conditions are depicted inside transitions rectangles. A transition t may fire *iff* there exists an assignable set in each input place of t and the assignment of values to variables by the assignable set satisfies the condition of t. Also, each outgoing arc from transition t to a place p' has a label of the form of $k'_1Y_1k'_2Y_2...$ where k'_i is a positive integer and Y_i is a n'-tuple of values, variables on the incoming arc labels of t or functions over the variables. Therefore, if t fires, the values of the labels on the outgoing arcs from t are determined by the assigned input tokens according to the output arc labels. New sets of tokens are generated and put into the output places of t.

In Fig. 3.3(a), the incoming arc to t from p_1, (p_1, t), has the label $2\langle x, y \rangle$ where x and y are variables. This means that two tokens each consisting of a pair of values are necessary in place p_1 for the firing of transition t. Here, since the following assignable sets $2\langle$"a","c"$\rangle$ in p_1 ("a" and "c" are strings here), $\langle$"a"$\rangle$ and $\langle$"c"$\rangle$ in p_2 and two tokens without values in p_3 satisfy the condition of t, $(x = z \wedge y = w)$, t can fire using these sets. Note that tokens without values are represented as black dots in the following figures. After the firing of t, new tokens are generated to the output places p_4 and p_5 using those token values. The marking after the firing of t is shown in Fig. 3.3(b). Note that "@" is a concatenation function of two strings. Thus a tuple of

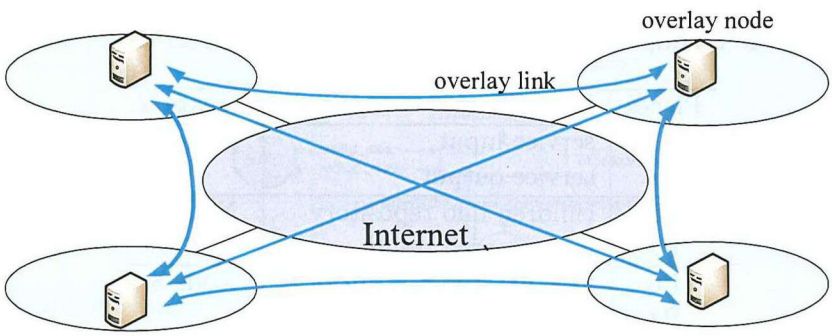


Figure 3.4: Network Architecture.

strings "aa", "c" and "a" is generated to p_4. The arc label "1", which means the delivery of one token without values, is omitted in the following figures.

3.4 An Overlay Network

We assume that a network is a logical overlay network where each server can communicate with any other server over the unicast link. This network is composed of two kinds of servers: servers which can store sensing data and servers which can execute service components. The processing power is different with respect to each server and the bandwidth of data transferring is also different with respect to each link.

The network is modeled as a directed complete graph G defined as $G = (N, L)$ such that N is a server set and L is a link set defined as $L = N \times N$.

3.5 Service Execution on an Overlay Network and Optimization Algorithm

Generally, in a service to collect and process sensing data stored at distributed places on an overlay network, a server to store the sensing data may be different from a server to process it. If it is actually different, data transferring between these servers is needed. For example, when transition $T3$ in Fig.3.2 is allocated in server i and output place $P5$ of transition $T3$ is allocated in server j, movie data which is a result of transition $T3$ is transferred on the overlay link from server i to server j. If the link bandwidth

Table 3.1: Example of Place Location Table for Child Tracking Service.

Places	Role	Location
P1	service input	Node 1
P4	service output	
P5	children info repository	Node 2
P2	index search result	
P6	location info repository A	Node 3
P3	repository search result	
P7	movie repository B	Node 4

between server i and server j is insufficient, that may lead that the data transferring becomes a bottleneck of the process. If transition $T3$ is allocated in server j as well as place $P5$, the data transferring between transition $T3$ and any other places except place $P5$ may be needed and become a new bottleneck though there is no data transferring from transition $T3$ to place $P5$. In terms of not only a link bandwidth but also the processing power of each server, we need to consider this bottleneck problem as well. We assume that the processing time of each transition is different since each server has a different processing power respectively. Under this assumption, the performance of server i itself may be insufficient and become a bottleneck depending on an allocation.

As mentioned above, it is desirable that the allocation of places and transitions to servers should be determined by considering this problem to process requests of a service totally fast. In this chapter, we call the allocation of places and transitions of a service description to servers as a distributed execution sequence. Our method aims at obtaining the distributed execution sequence to maximize the number of processed requests per unit of time (called as throughput of the service hereafter).

3.5.1 Deriving Distributed Version of Service

In this section, we briefly introduce the method proposed in [13] to derive a distributed version of a given service. The platform presented in this thesis is based on this method. We briefly explain the algorithm for deriving

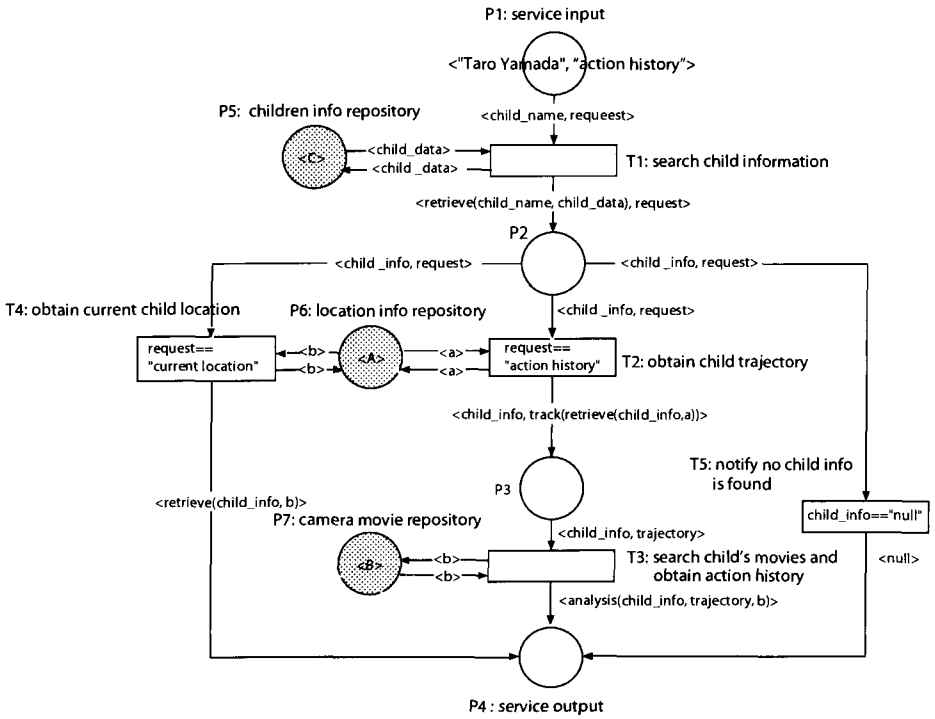


Figure 3.5: Child Tracking Service written in a Coloured Petri Net.

a distributed version of a given service. The algorithm takes as inputs a colored Petri net description of a service and an allocation table of places into overlay nodes (called *place location table*), and outputs a distributed version of the service, which is a set of coloured Petri nets that contain communicating behavior between the nodes. Each obtained coloured Petri net corresponds to the behavior description of an overlay node.

We assume that two places with a common name "$X_{u.ij}$" (X is used in the derivation algorithm and is α or γ [13]) in the coloured Petri nets of two different nodes where i and j represent the end points (send and receive buffers) of a reliable communication channel from node i to node j. If a token is put on place "$X_{u.ij}$" at node i, the token is eventually removed and put onto the same place "$X_{u.ij}$" at node j. These places are called *communication places*, and are like "fusion places" in coloured Petri nets[1]. Note that u means that these communication places are used with

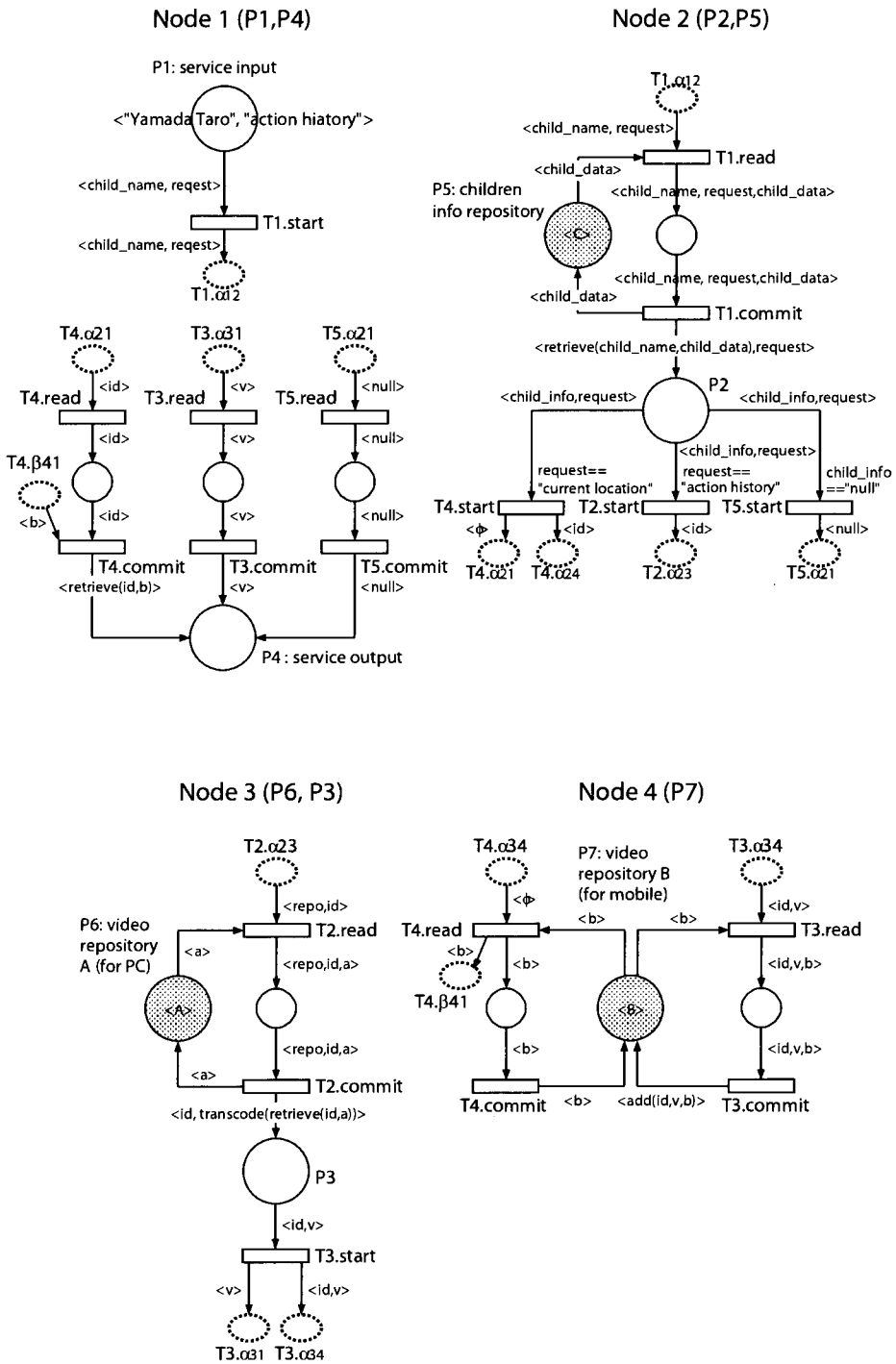


Figure 3.6: Distributed Child Tracking Service of Fig. 3.1 under Place Location of Table 3.1.

respect to the execution of transition T_u of the service. In the following figures, communication places are represented as dotted circles with their names inside. Also, in a distributed version of a given service, we introduce a reserved symbol denoted by ϕ, used in tokens for notification purpose only.

We show an another example of a service description in Fig.3.5. This service presents a tracking service for children and their parents. Fig. 3.6 shows a distributed version of the service of Fig. 3.5 on four overlay nodes based on the place location table in Table 3.1. The reader may refer to [50] for a complete description of the algorithm. Here, we provide a simple example that demonstrates how nodes can collaborate for providing a given service in a distributed environment and how such cooperative behavior is described using coloured Petri nets with communication places. In the algorithm, one of the overlay nodes that has input places of a transition is selected as the node that starts the execution of that transition (such a node is called the *primary node* of the transition). In Fig. 3.6, node 1 is selected as the primary node of $T1$ since it has place $P1$. Whenever the input place $P1$ of $T1$ receives a token (a request with keywords), node 1 instantaneously sends this token to node 2 which has place $P5$ through the communication place "$T1.\alpha_{12}$" (it is worth noting that both nodes 1 and 2 include the communication place "$T1.\alpha_{12}$"). Afterwards, at node 2, the retrieval from video index $P5$ is done and the result $retrieve(keys, index)$ is generated as a token into place $P2$ at node 2. This simulates the behavior of $T1$. Place $P2$ represents a choice of $T2$, $T4$ or $T5$ based on the token value of variable '*repo*" in $P2$, and this value has been determined as the result of the previous computation $retrieve(keys, index)$. The reader may verify that the coloured Petri nets of Fig. 3.6 simulate/realize the service of Fig. 3.2.

3.5.2 Place Allocation to Overlay Nodes

One of the important features of our methodology is to decouple service descriptions from overlay network specifications. Accordingly, developers are allowed to write services without being aware of the actual locations of data. That is, they can refer to and update tokens in any place from any

transition in writing services. To accomplish this, the method given in [13] provides a way for distributing the given service into overlay nodes knowing an allocation of the places to the overlay nodes. However, it is clear that the performance of a service is affected by this allocation of places. For example, allocating a frequently-accessed database into an overlay node with limited computation power usually leads to a performance bottleneck. Accordingly, in [13], before distributing a given service into overlay nodes, an optimal allocation of places into the overlay nodes is determined using a given optimization model. The model takes into account the computation power of the overlay nodes, the network capability, and the execution orderings of subtasks in given services and simulation experiments were conducted to see the effectiveness of the optimization. The reader may refer to [13] for a detailed description of the optimization model.

The platform presented in this thesis implements this optimization model. In addition, optimization is done using collected information about the performance of the overlay nodes and the network. This helps in optimizing the performance of a distributed service as will be demonstrated in Section 3.6.

3.5.3 An Example of a Service Execution based on a Distributed Execution Sequence

A distributed execution sequence is a map to a set of severs N denoted as $alc : P \cup T \rightarrow N$ where P is a set of places and T is a set of transitions. We assume that a set of input places of transition t is denoted as $\bullet t$ and a set of output places is $t\bullet$. We also assume that a set of input transitions of place p is denoted as $\bullet p$ and a set of output transitions is denoted as $p\bullet$ as well. A service execution based on a distributed execution sequence is described below in details.

An execution of transition t is conducted by the following three steps.

(I) In each input place $p_x \in \bullet t$ of transition t, colored tokens in the place p_x are transferred from server $alc(p_x)$ to server $arc(t)$. These transfers can be conducted separately.

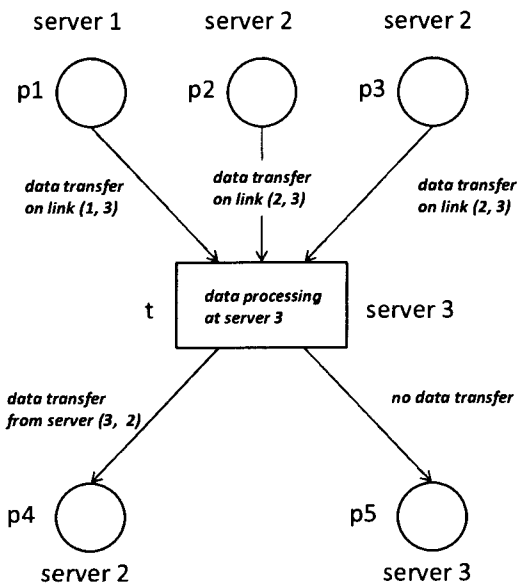


Figure 3.7: Example of a transition execution.

(II) Server $alc(t)$ executes transition t immediately after all coloured tokens have been transferred to server $arc(t)$.

(III) As to each output place $p_y \in t\bullet$ of transition t, coloured tokens to be generated in place p_y are transferred from server $alc(t)$ to server $alc(p_y)$. These transfers can be conducted separately as well as Step (I).

If requests arrive at the service continuously, each step of Step(I) thru Step (III) in the transition t mentioned above can be processed like pipeline processing. Among Step (I) thru Step (III), the step for which it takes most delay to process one request becomes the bottleneck of the execution of transition t and the number of requests processed per unit of time in transition t is determined by the bottleneck. Given this factor, we assume that the throughput of transition t (called $th(t)$) is denoted as the minimum value of the number of requests processed per unit of time among Step (I) thru Step (III) of transition t.

For example, Fig.3.7 shows an example of a distributed execution of tran-

sition t. In Fig.3.7, place $p1$ is allocated to server1, place $p2$, place $p3$, and place $p4$ are allocated to server2, and place $p5$ and transition t are allocated to server3. On the execution of transition t, in the Step (I), the following two data transfers should be conducted (labeled Substep (a) and (b)). At first, in the Substep (a), a coloured token in place $p1$ is transferred from server1 to server3. Next, in the Substep (b), coloured tokens in place $p2$ and place $p3$ are transferred from server2 to server3. After all coloured tokens have been transferred to server3, in the Step (II), transition t is executed on server3, and coloured tokens of place $p4$ and place $p5$ are generated (labeled Substep (c)). Finally, in the Step (III), labeled Substep (d), the coloured token of place $p4$ is transferred from server3 to server2. However, the coloured token of place $p5$ is transferred between servers since transition t and place $p5$ are allocated to the same server3.

The throughput of transition t in Fig.3.7 is the minimum value of the number of requests processed per unit of time among Substep (a) thru Substep (d) of the transition. We assume that the throughput of the entire service (called $th(S)$) is defined as the sum of the throughput of all input transitions in place p_{out}. The goal of our optimization algorithm is to maximize the $th(S)$.

3.5.4 Problem Definition

In this section, we define the optimization problem to maximize a throughput of a service.

The throughput of Step(I) and Step (III) mentioned in the previous section is limited depending on a link bandwidth between servers and amount of data transferring between the servers. To represent the limitation, we assume that maximal throughput $TH_{msg}(u, v, i, j)$ is given, and this value depends on the link bandwidth from server i to server j $(i, j \in N)$ and amount of data through arc (u, v) $(u, v \in P \cup T)$ where N is a set of servers, P is a set of places and T is a set of transitions. On the other hand, the throughput of Step(II) is limited depending on a processing power of a server and calculation amount of a component. To represent the limitation, we also assume that maximal throughput $TH_{exec}(t, i)$ is given depending on

the processing power of server i and calculation amount of transition t. Additionally, if multiple transitions refer the same repository(place), each throughput of transitions referring the repository is limited depending on the processing power of the server where the repository is allocated. To represent the limitation, we assume that maximal throughput $TH_{db}(r,i)$ of repository r ($r \in P$) is given depending on the processing power of server i. It seems unreasonable that the server where a repository is allocated is different from the server where the transition to refer the repository is allocated, since all data of the repository need to be transferred between servers. Hence, in this optimization problem, we assume that each transition to refer a repository should be allocated to the server where the repository is allocated. Essentially, this maximal throughput is changeable depending on allocations, for example, in the case when multiple transitions are allocate to the same server.

In our proposed method, we reduce this problem to a linear programming problem. In the linear programming problem, given service description S written by a coloured Petri net, network graph G where $G = (N, L)$ and the following sets of maximal throughput: $TH_{exec}(t,i)$ $(\forall t \in T, \forall i \in N)$, $TH_{msg}(u,v,i,j)$ $(\forall u,v \in P \cup T, \forall i,j \in N)$ and $TH_{db}(r,i)$ $(\forall r \in P, \forall i \in N)$, we introduce Boolean variable $alc(v,i)$ which is 1 if v is allocated to i and 0 if v is not allocated to represent a distributed execution sequence $alc : P \cup T \rightarrow N$. Furthermore, for all arcs $(u,v) \in (P \times T) \cup (T \times P)$ and all links $(i,j) \in L$, we also introduce Boolean variable $msg(u,v,i,j)$ which is 1 if u is allocated to i and v is allocated to j, and 0 if not allocated.

3.5.5 A Linear Programming Problem for Service Optimization Problem

[Objective Function]

The goal of this problem is to maximize throughput $th(S)$ of a service. The throughput $th(S)$ of a service means the number of coloured tokens per unit of time which is put to output place p_{out} of the service. The objective

Function is defined as the following formula.

$$max\ th(S) = \sum_{t \in {}^\bullet p_{out}} th(t) \tag{3.1}$$

[Constraints for throughput]

The throughput $th(t)$ of each transition $t \in T$ needs to satisfy the following formula.

$$th(t) \leq min(X \cup Y \cup Z) \tag{3.2}$$

Let $min(A)$ be a function to obtain the minimal element from a set A. X, Y and Z is denoted as the following.

$$X = \{TH_{msg}(p,t,i,j) \mid msg(p,t,i,j) = 1 \wedge p \in {}^\bullet t \wedge (i,j) \in L\} \tag{3.3}$$

$$Y = \{TH_{exec}(t,i) \mid alc(t,i) = 1 \wedge i \in N\} \tag{3.4}$$

$$Z = \{TH_{msg}(t,p,i,j) \mid msg(t,p,i,j) = 1 \wedge p \in t^\bullet \wedge (i,j) \in L\} \tag{3.5}$$

(3.3)-(3.5) mean constraints for maximal throughput in Step(I), Step(II) and Step(III) respectively. (3.2) describes $th(t)$ is denoted as the minimal throughput of Step (I) thru Step (III). Since the objective function aims at maximizing the value of the throughput $th(S)$, (3.2)-(3.5) can be reduced to the following linear equations:

$$th(t) - TH_{exec}(t,i) - C \cdot (1 - alc(t,i)) \leq 0 \tag{3.6}$$

where $\forall t \in T, \forall i \in N$ and

$$th(t) - TH_{msg}(p,t,i,j) - C \cdot (1 - msg(p,t,i,j)) \leq 0 \tag{3.7}$$

$$th(t) - TH_{msg}(t,p,i,j) - C \cdot (1 - msg(t,p,i,j)) \leq 0 \tag{3.8}$$

where $\forall t \in T, \forall p \in P, \forall (i,j) \in L$. It is assumed that C is a constant where it is much greater than $TH_{msg}(p,t,i,j)$ and $TH_{msg}(t,p,i,j)$. Consequently, these constraints is true if $alc(t,i)$ or $msg(u,v,i,j)$ equals 0. (3.6) corresponds to (3.4) and (3.7) and (3.8) correspond to (3.3) and (3.5) respectively. (3.6) represents that $th(t) - TH_{exec}(t,i) \leq 0$ is true only if transition t is allocated to server i. (3.7) and (3.8) represent the same as well as (3.6).

On the other hand, focusing on places, input and output of coloured tokens to a place can be regarded as a data flow. Hence, the following formula where $\forall p \in P \setminus \{p_{in}, p_{out}$ is defined.

$$\sum_{t \in \bullet p} th(t) - \sum_{t' \in p\bullet} th(t') \geq 0 \tag{3.9}$$

[Constraints for transitions referring to repositories]

Let R be a set of places denoted as repositories where $R \subset P$ and let $T'(r)$ be a set of transitions referring to repository r where $r \in R$. Since these transitions should be allocated to the same server as mentioned in Section 3.5.4, the following constraint where $\forall r \in R, \forall i \in N, \forall t \in T'(r)$ can be described.

$$alc(t,i) - alc(r,i) = 0 \tag{3.10}$$

Furthermore, as to the throughput of transitions to refer repositories, the following constraint where $\forall r \in R, \forall i \in N$ can be described.

$$\sum_{t \in T'(r)} th(t) - TH_{db}(r,i) - C \cdot (1 - alc(r,i)) \leq 0 \tag{3.11}$$

We assume that C is constant where it is much greater than $TH_{db}(r,i)$ as well as (3.6).

[Constraints for communications between servers]

A constraint for $msg(u,v,i,j)$ is defined as the following formula where $\forall (u,v) \in (P \times T) \cup (T \times P), (i,j) \in L$.

$$msg(u,v,i,j) = alc(u,i) \cdot alc(v,j) \tag{3.12}$$

(3.12) means that $msg(u,v,i,j)$ equals 1 only if u is allocated to i and v is alocated to j. (3.12) can be reduced to the following linear inequalities where $\forall (u,v) \in (P \times T) \cup (T \times P), \forall (i,j) \in L$ since variable $alc(u,i)$ and variable $alc(v,i)$ are Boolean variables.

$$alc(u,i) - msg(u,v,i,j) \geq 0 \tag{3.13}$$

$$alc(v,j) - msg(u,v,i,j) \geq 0 \tag{3.14}$$

$$alc(u,i) + alc(v,j) - msg(u,v,i,j) \leq 1 \tag{3.15}$$

Table 3.2: Experimental Settings.

Parameters	Transportation analysis	Floating car data analysis
network	PlanetLab	1000BASE Ethernet
overlay	full-meshed	full-meshed
# of servers	30	5
# of requests	300	815
	60 (for traffic forecasting)	500 (for traffic estimation)
	120 (for movie searching in $T8$)	300 (for OD time estimation)
	120 (for movie searching in $T9$)	15 (for traffic forecasting)

[Constraints for allocations]

The following formula where $\forall v \in P \in T$ should be described since every place and transition is certainly allocated to a server.

$$\sum_{i \in N} alc(v, i) = 1 \tag{3.16}$$

As mentioned above, the optimization problem of a service is reduced to a linear programming problem(ILP). We can obtain an optimal distributed execution sequence by solving the problem. Though an integer linear programming problem belongs to a NP-hard problem, we expect that this problem can be solve in realistic time since it is assumed that a service is composed of few dozens of places and transitions. Furthermore, it is comparatively easy to obtain a quasi-optimum solution because many heuristic algorithms to solve are proposed. Hence, it seems that our method can apply practical services.

3.6 Experiments

In this section, for two kinds of services: transportation analysis services in Fig.3.2 and floating car data analysis services in Fig.3.8, we conduct evaluation of the distributed execution sequence derived by the proposed method and improvement on performance of the service using the proposed platform.

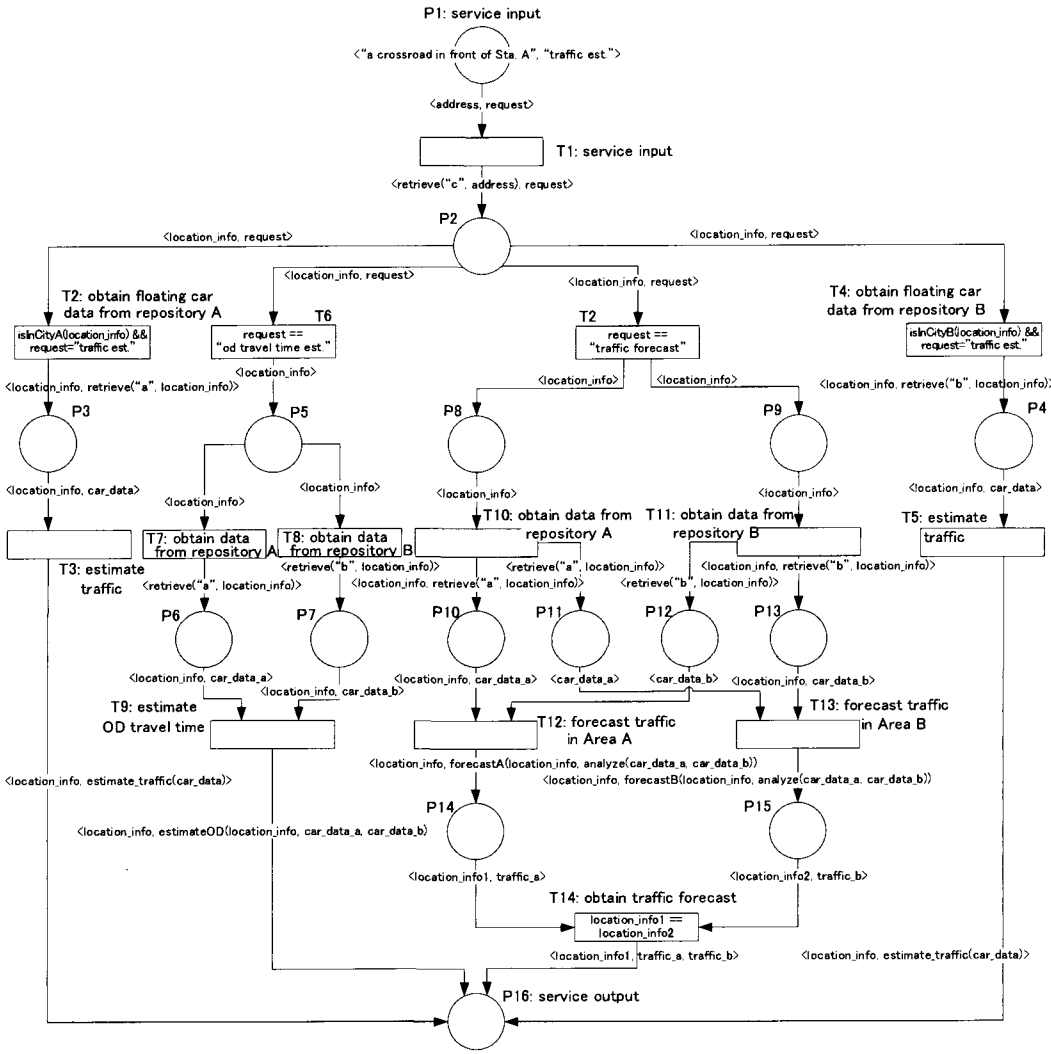


Figure 3.8: Floating Car Data Analysis Services.

The floating car data analysis services in Fig.3.8 provides traffic information such as traffic estimation at a crossroad, estimating origin-destination travel time and traffic forecasting for car navigation systems by analyzing floating car data. Each vehicle in west side and east side of Umeda in Osaka sends floating car data to repository A and repository B respectively. In this service, we assume that the average of the number of vehicle in these areas

is about 7,500, 5% of vehicles are floating cars and 67% of vehicles equip sets of a car navigation system.

Table 3.2 shows experimental settings. In this experiment, we use a network testbed PlanetLab [51] which is composed of worldwide servers on the Internet for the transportation analysis services in Fig.3.2. We build a full-mesh overlay network over TCP composed of 30 servers on PlanetLab which belong to different domains each other. On the other hand, we use 1000BASE Ethernet for the floating car data analysis services in 3.8. We build a full-mesh overlay network composed of 5 servers which belong to the same domain. However, pseudo-packet delay generator should work at each node to represent a network with high latency.

Moreover, data repositories (Ex. place $P13$ and place $P14$ in Fig.3.2) are allocated to fixed servers since these places depend geographically. we execute the two kinds of services based on not only a distributed execution sequence derived by the proposed method but also random distributed execution sequences for comparison. On deriving a distributed execution sequence by the proposed method, we measure CPU power of each server and available bandwidth of each link between servers using the function of the proposed platform. Based on these values, we give the max throughput that is a input of the proposed algorithm.

To evaluate the performance of a distributed execution sequence designed by the proposed method, we compare it exhaustively with distributed execution sequences generated at random as many as possible. In this experiment, we generate 100 distributed execution sequences at random in Fig.3.2 and Fig.3.8 respectively. 300 requests at each distributed execution sequence in Fig.3.2 are generated. These requests are composed of 60 requests for traffic forecasting, 120 requests for movie searching in transition $T8$ and 120 requests for movie searching in transition $T9$. On the contrary, 815 requests at each distributed execution sequence in Fig.3.8 are generated. These requests are composed of 500 requests for traffic estimation at a crossroad, 300 requests for estimating origin-destination travel time and 15 requests for traffic forecasting. We input these requests in the random order. Under these settings, we measure throughput of the service at each distributed

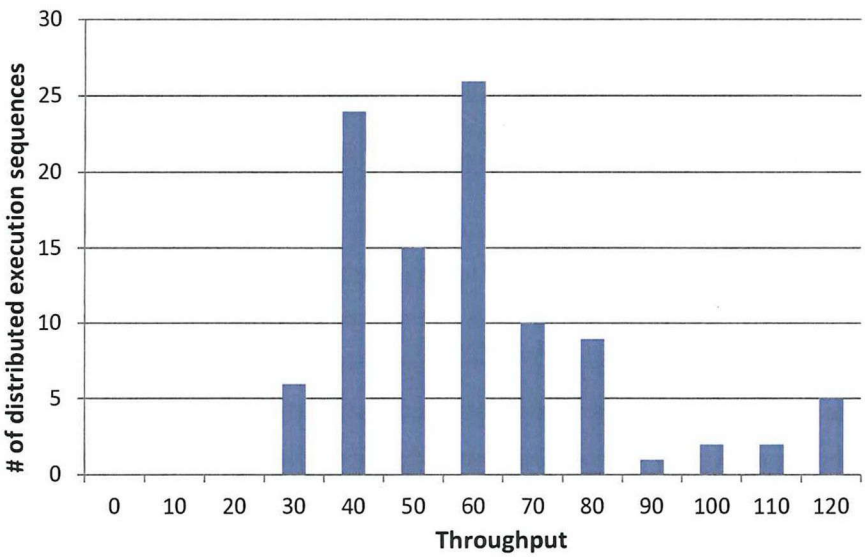
Table 3.3: Throughput of Services Derived by the Proposed Method.

Services	Throughput
Transportation analysis services	114
Floating car data analysis services	308

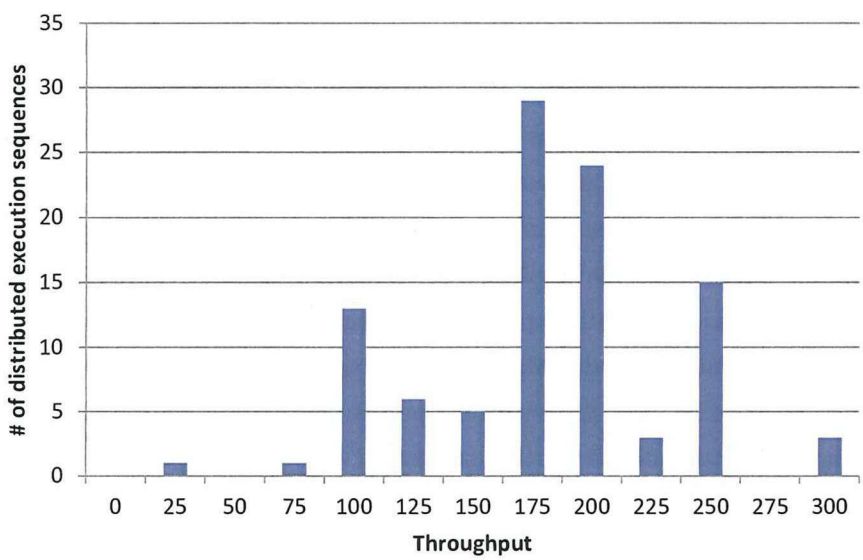
execution sequence.

3.6.1 Throughput of a Service

Table 3.3 shows the throughput of each service derived by the proposed method and Fig.3.9 shows a frequency distribution table of the average number of requests completed to process per 100 seconds defined as throughput of the service for random distributed execution sequences. In this table, the x-axis represents the throughput and the y-axis represents the number of distributed execution sequences whose throughput is $(x-10, x]$. As to the transportation analysis services, the maximal throughput is 116.2 and the minimal throughput is 23.6 in (a) of Fig.3.9. This represents that the throughput changes quite much depending on distributed execution sequences. On the other hand, the throughput of the distributed execution sequence designed by the proposed method is 114. This throughput is the 3rd highest of the throughput of random distributed execution sequences. Correspondingly, in the floating car data analysis services, the maximal throughput is 292 and the minimal throughput is 21.0 in (b) of Fig.3.9. The throughput of the distributed execution sequence designed by the proposed method is 308. This throughput is the highest of the throughput of random distributed execution sequences. We observe that the proposed method can derive distributed execution sequences which have high performance. As reasons why the distributed execution sequence by the proposed method does not perform the best in the transportation analysis services, we guess that the distributed execution sequence is not optimal any more on execution since server and network utilization are always changing on the PlanetLab, and we also guess that the network model may be insufficient to describe real networks since a portion of links in a real network can be



(a) Transportation Analysis Services.



(b) Floating Car Data Analysis Services.

Figure 3.9: Frequency Distribution of Throughput.

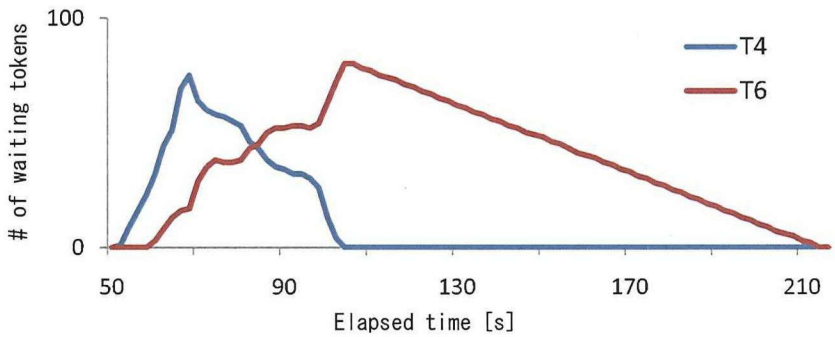


Figure 3.10: The Number of Waiting Tokens at $T4$ and $T6$

conflicted although a network is modeled as a full-mesh overlay network in the proposed method.

When we optimize services, workloads of the network and servers during the entire processing period of the services should be considered. Therefore, load statistics during the period should be given to the proposed method as network information. The proposed method intends to optimize one service but services can balance each other on a network when each service keeps its performance not to exceed constraints of workloads. Additionally, if a network provides functions of load balancing, the proposed method can use these functions as a constraint. If every service uses our platform proposed in Chapter 4, our platform can measure the performance of all services. Therefore, our platform can be used as a framework which provides the functions of load balancing, giving constraints for max throughputs of servers and a network.

3.6.2 Bottlenecks of a Service

Next, we show an example of a process to discover a bottleneck of a service which causes determines We focus on the number of waiting tokens at transitions since waiting tokens may occur at bottlenecks. Fig. 3.10 shows the number of waiting tokens at transition $T1$ and transition $T2$ in a distributed execution sequence. We can observe that servers where the transitions are allocated are overloaded since waiting tokens occur at both of the transi-

tions. When new tokens don't go into to the transitions, the transition where the decreased number of tokens is higher than another transition performs better. Hence, the bottleneck is not at transition $T4$ but transition $T6$ because the decreased number of tokens in transition $T4$ is higher than transition $T6$. It may be possible to improve on the throughput of the service if $T6$ is reallocated to more appropriate server. As mentioned above, we can discover bottlenecks easily using the monitoring function of the proposed platform.

3.7 Conclusion

In this chapter, we have proposed a method to derive execution sequences of a given application-level service that is executed by cooperative servers on overlay networks. The proposed method assumes that a service consists of service components, and it can derive optimal allocation of components that does not overload network links and servers. Using the platform mentioned in Chapter 4, services can be installed and executed easily on real networks. The platform can measure network loads and CPU loads continuously, and can adaptively determine new allocation optimizing the system load in the environment. We have conducted experiments on PlanetLab to validate our method. We have confirmed that the proposed method could derive efficient execution sequences and they could achieve higher throughput than the other sequences.

Chapter 4

Design and Development of Service Execution Platform for Overlay Networks

4.1 Introduction

In this chapter, we design and develop a service execution platform for supporting the development of overlay services. Using the platform, services can be installed and executed easily on real networks. Given network information, this platform can build overlay network automatically. Then, the platform deploy the distributed services which is derived by the design method to servers by SSH. On executing the derived service, this platform can monitor network loads and CPU loads continuously in order to confirm the performance of the service by a service designer.

4.2 Service Execution Platform

Our platform mainly consists of the following functional components; (i) *distributed service generator* that derives distributed versions of services, (ii) *resource optimizer* that is the part of the distributed service generator and suggests the "optimal" allocation of places for better performance of the services, (iii) *service executer* that interprets distributed service descriptions written in coloured Petri nets and executes the services on overlay nodes, and (iv) *resource monitor* that monitors overlay nodes and link status.

4.2.1 Distributed Service Generator and Resource Optimizer

The distributed service generator takes as an input a service written using CPN Tools, coded according to CPN Tools XML format, and a description file of the given service parsed using our dedicated/special XML parser designed for this purpose. Moreover, the generator also takes inputs regarding the overlay network such as the number of overlay nodes and the nodes identifiers, IP addresses and ports.

The service description and the overlay service information are given to the resource optimizer which generates, from the give information, an allocation of places, represented as a place location table, that optimizes the performance of the service. Optimizing the performance is done using the integer linear programing (ILP) model given [13]. We use CPLEX to solve the optimization problem and obtain a place location table that achieves the best performance in terms of the considered objective functions. Based on the derived place location table, the corresponding distributed service description is then derived by the distributed service generator. However, due to network uncertainty, the optimized place allocation does not always provide a good performance. To obtain better performance, we provide more information about the overlay networks' status, such as the overlay link status (delay and bandwidth) and the overlay node status (CPU load) which are usually difficult to predict in advance. Thus, we may need to run again the optimization problem and evaluate, by monitoring the execution of the service, the obtained results using the real network . We show how our platform achieves this goal in Section 3.6.

4.2.2 Service Executer

The architecture of the serive executer function is shown in Fig. 4.1. This function is realized by the cooperation of service access point module, a controller, and a set of overlay nodes. The controller consists of an interpreter of the distributed services, written using CPN tools format, and a system controller that handles the exchange of data among the overlay nodes and the execution of subtasks according to the commands provided by the interpreter. The service access point provides users with Web interfaces for

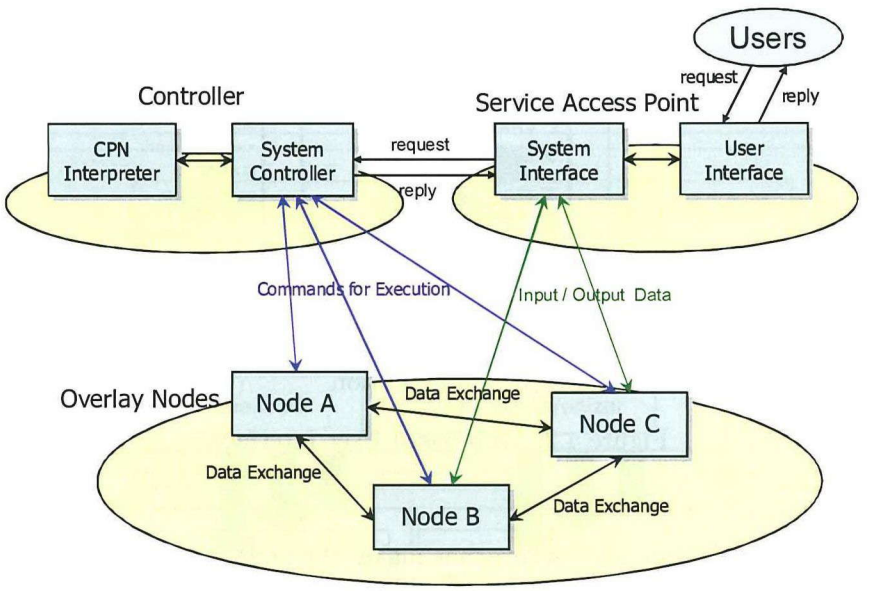


Figure 4.1: Overview of Service Executer.

accessing services and also has a system interface module that allows the exchange of input/output data between users and services.

The system interface receives requests from users through the Web-based user interface in Fig.4.2. Then, it requests the CPN interpreter to put a corresponding token into the service. This request is passed through the system controller. The system interface also requests the controller to put corresponding data onto the overlay nodes. After executing the service, the system interface receives the output of the service and replies back to the system user. The output is provided to the user as a URL.

The *system controller* is designed as shown in Fig. 4.3. The controller manages the execution of at the overlay nodes by receiving commands from the CPN interpreter. When the system starts, it initializes the CPN interpreter and registers the distributed service at the interpreter. Also it sends the (compiled) subtask program codes of the service to the overlay nodes that execute these subtasks. The interpreter also sends to every overlay

(a) Before execution (b) On execution (c) After execution

Figure 4.2: Web-based User Interface.

node an initialization shell script that makes the node ready to execute the service by establishing overlay links with other nodes. Managing connections between the system controller and the overlay nodes is done by the *Node Maintainer*. Moreover, the association of overlay nodes of a coloured Petri net service with the actual overlay nodes is done by a *Node Mapper*. Every connection is established using the Secure SHell (SSH) program and the inter-process communication port of the system controller is redirected to the standard I/O of the SSH program.

Message Processor is a message handler that communicates the control messages between the system controller and the overlay nodes. Finally, the *Data Synchronizer* collects the log files recorded at every overlay node using the SCP or the RSYNC programs.

The system controller has an *Event Monitor* that intermediates the CPN interpreter and the overlay nodes. When the CPN interpreter finds an executable transition, the transition is fired and the programs that correspond to its subtasks are executed at the corresponding overlay nodes. Here, we note that in our model the firing of transitions is not instantaneous since the subtasks associated with transitions (represented as labels of output arcs) may require some execution time. For example, in Fig. 3.6, *T2.commit* of node 3 has a function "$transcode(retrieve(id, a))$" on its output arc. The function represents a subtask to downsize the video retrieved from repository

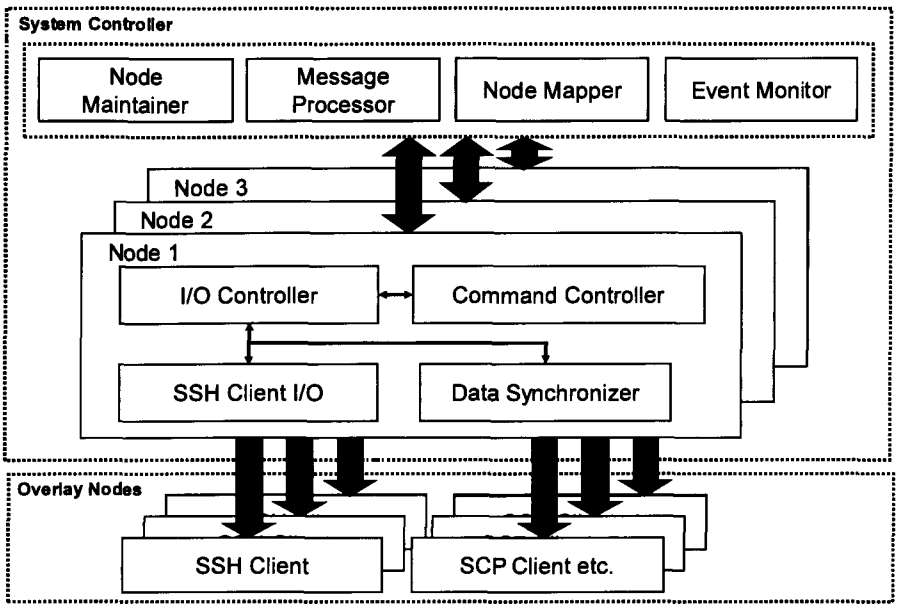


Figure 4.3: System Controller.

A and this subtask may take considerable execution time for transcoding. Similarly, the transmission of large data between overlay nodes (represented as communication places) may require a transmission delay. The event monitor observes the execution of the programs and the transmissions of data, and upon completion, the monitor notifies to the CPN interpreter. The CPN interpreter, after the firing of transitions, delays the generation of tokens to the output places and also delays moving tokens between communication places of different nodes until receiving the notifications from the event monitor. We note that we have developed a dedicated/special CPN interpreter although there are several simulators that can deal with high level Petri nets. This is done to take care of aspects discussed above. In our case, the CPN interpreter needs to interact with the system controller, which controls overall execution of services on overlay nodes, and also needs to deal with multiple Petri nets representing the behavior of overlay nodes.

Table 4.1: System Controller's Commands for Overlay Nodes.

command	receiver node's behavior
connect *node*	send a connection request to *node*
accept *node*	wait for and accepts a connection request from *node*
disconnect *node*	disconnect the connection with *node*
suspend	suspend execution
resume	resume execution
transmit *file* *node*	send *file* to *node*
execute *prog*	execute *prog*
sync	execute clock sync. module to synchronize the nodes' clocks with each other
exit	quit from service

4.2.3 Overlay Node Control and Network Management

The overlay nodes execute the programs, which correspond to the subtasks of services including those for exchanging data between nodes, according to the commands given by system controller. These commands are listed in Table 4.1.

We note that since we allow parallelism in the behavior of an overlay node, while transmitting data from an overlay node, another transmission request with the same destination node may occur. we may delay the latter request till the completion of the current transmission. However, in this case, the response time of the latter request may be long, especially when dealing with large-size files such as video files.

To overcome this problem, we have implemented a multiplexer that allows the sharing of an overlay link among multiple transmissions of data with designated occupancy ratios. Here, the occupancy ratio of one transmission is set as the ratio of the transmission's throughput to the overlay link throughput. To realize parallel transmission of data, the system segments the data into fixed size pieces. Fig. 4.4 shows an example where three data transmissions are transferred, in parallel, from node A to node B with

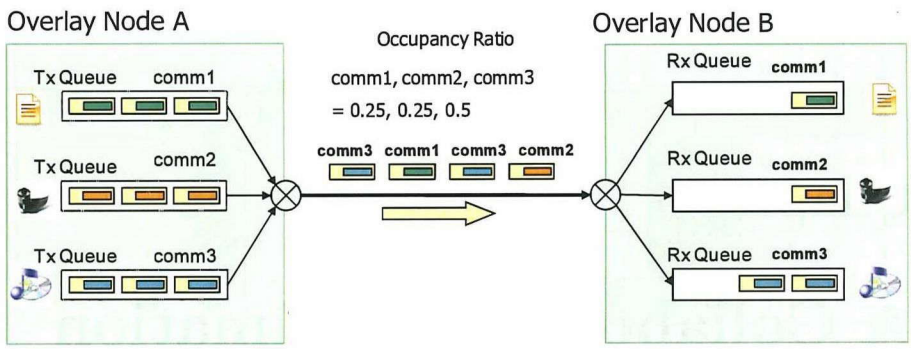


Figure 4.4: Overlay Link Control.

occupancy ratios 0.25, 0.25 and 0.5, respectively.

4.3 Conclusion

In this chapter, we have designed and developed a service execution platform. Using the platform, services can be installed and executed easily on real networks. The platform can measure network loads and CPU loads continuously, and can adaptively determine new allocation optimizing the system load in the environment.

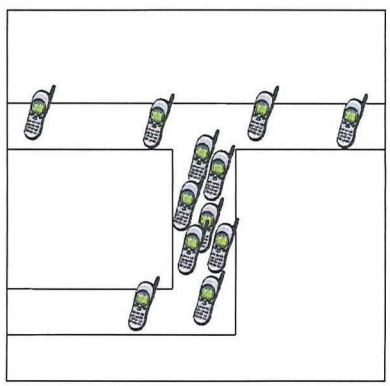
Chapter 5

A Collaborative Estimation Protocol of Distribution for Mobile Wireless Nodes

5.1 Introduction

In this chapter, we propose a method for mobile wireless nodes, which are pedestrians, to estimate the distribution of mobile nodes in their surroundings. In the proposed method, each node is assumed to know its location roughly (*i.e.* within some error range) and to maintain a density map covering its surroundings. This map is updated when a node receives a density map from a neighboring node. Each node also updates its density map in a timely fashion by estimating the change of node distribution over time due to node mobility.

The rest of this chapter is organized as follows. Section 5.2 presents an algorithm for self-estimation of neighborhood distribution. Section 5.3 describes experimental results for evaluation of the proposed algorithm. Section 5.4 shows the discussion about the proposed algorithm Finally, this chapter is concluded in Section 5.5.



(a) Positions of nodes

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0.3	0.4	0.4	0.5	0.5	0.2	0.1	0.1
0	0	0	1.5	1.0	0	0	0
0	0	0	1.4	1.3	0	0	0
0	0	0	1.0	1.0	0	0	0
0	0	0.4	0.6	0.6	0	0	0
0	0	0	0	0	0	0	0

(b) A density map

Figure 5.1: An Example of a Density Map

5.2 Self-Estimation of Neighborhood Distribution

5.2.1 Overview

We assume that each node i is equipped with a wireless device and knows its (rough) location through GPS or other technologies. We also assume that the region is divided into square cells with s(m) edge. Based on this cell representation of the geography, node i maintains a density map D_i, which represents the node density (i.e. the expected node numbers) in its surroundings. Concretely, D_i has $X_i \times Y_i$ elements and each element $d_{x,y}(1 \leq x \leq X_i, 1 \leq y \leq Y_i)$ represents the node density in the cell (x, y). We define the size X_i, Y_i and the location of the density map as node dependent values since each node may require its local view of the density map depending on applications. An example of a density map is shown in Fig. 5.1. We assume each node knows the maximum speed V_{max} of all the nodes to estimate the change of the node density in each cell by predicting movement of nodes. This is because the maximum speed may be estimated easier than the average speed, which largely depends on the time, the locations, the density and geometrical attributes such as the path's width.

Each node i executes the following procedures every t seconds.

1. Node i updates its density map D_i by using a given estimation function f. We assume a typical moving pattern in the target environment is

modeled into the estimation function. According to this model, $f(D_i)$ diffuses the density values in each cell toward its surrounding cells that are supposed to be reachable within a message exchange interval denoted by t. This represents the estimated movement of other nodes. We note that in $f(D_i)$, if $d_{x,y}$ is less than a certain threshold denoted by TH_d after updating, $d_{x,y}$ is set to zero. For TH_d, we set the value which is too small or too old as the density information and which is therefore not useful any longer.

2. Node i adds its presence information to D_i. To do this, firstly, node i obtains its area of presence (denoted by R_i) from the GPS or other measurement devices where R_i is the area which includes node i's true position. We represent R_i as a set of cells as follows;

$$R_i = \{(x_{i1}, y_{i1}), (x_{i2}, y_{i2}), \ldots, (x_{in}, y_{in})\}$$

where n is the number of cells included in the area of presence. Thus the density value in each cell of R_i is $1/n$. Secondly, this value is added to the density value of each cell in the density map D_i. This procedure is executed only when the elapsed time since node i records R_i becomes longer than a certain Δt_i seconds. For Δt_i, we set the expected time for the density $1/n$ added to each cell to be less than a certain threshold (denoted by ε) due to the estimation function. Hence, Δt_i should be set according to the estimation function.

3. Node i sends D_i to its neighbors.

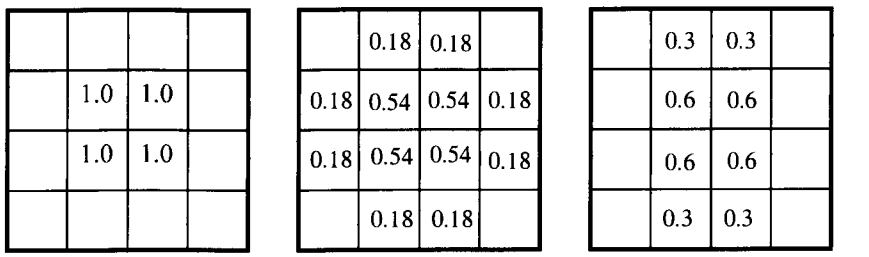
4. Node i updates D_i when i receives D_j from neighboring node j.

We explain the details of these procedures in the following section.

5.2.2 Algorithm

Estimation Function

Density maps are updated by the estimation function f, which is given beforehand. Typical movement patterns in the target region and/or the target nodes are modeled in the estimation function. Here, we describe (i)



(a) Before updating (b) After 1 iteration (Diffuse; α=0.9) (c) After 1 iteration (Limited diffuse;α=0.9)

Figure 5.2: Update by Estimation Function

```
for(step=0; step<floor(t*Vmax/s); step++){
 D'_i=D_i;
 foreach (d_(x,y) in D_i){
  d'_(x,y)=0.2*d_(x,y)*alpha;
  d'_(x-1,y)=d_(x-1,y)+0.2*d_(x,y)*alpha;
  d'_(x,y-1)=d_(x,y-1)+0.2*d_(x,y)*alpha;
  d'_(x+1,y)=d_(x+1,y)+0.2*d_(x,y)*alpha;
  d'_(x,y+1)=d_(x,y+1)+0.2*d_(x,y)*alpha;
  }
 D_i=D'_i;
}
return D_i;
```

Figure 5.3: Diffuse Estimation Function

the diffuse estimation function, (ii) the limited diffuse estimation function, and (iii) the hybrid estimation function as examples of typical movement patterns and their estimation functions.

Diffuse Estimation Function. When the maximum speed of nodes is the only known fact, there is a possibility that each node moves toward any directions in the region. Thus, the diffuse estimation function divides density values in each cell to its neighboring cells which have a shared edge with the cell. A weight $\alpha(0 < \alpha < 1)$ is considered when a density value is divided so that aging of information can be regarded. Because the edge size of a cell is s(m) and updates are repeated every t seconds, the diffuse estimation function iterates this procedure $\lfloor t * V_{max}/s \rfloor$ times. Fig. 5.2(b) and Fig. 5.3 show an example of the update by the diffuse estimation function and its

pseudo-code, respectively.

In this function, Δt_i is determined based on k which satisfies the following condition:

$$\frac{\alpha^k}{2k^2+2k+1} \leq \varepsilon \tag{5.1}$$

Here, k is the number of iterations by the diffuse estimation function. The left part in the above condition approximately denotes a density value in one cell after k steps, starting from a single cell of which a density value is 1. The denominator is the number of cells and the numerator means the freshness of the latest recorded area of presence. Each iteration is executed once in s/V_{max} seconds. Therefore,

$$\Delta t_i = \frac{k * s}{V_{max}}. \tag{5.2}$$

Limited Diffuse Estimation Function. There are actually movable areas and unmovable areas if pedestrians walk on roads. Here, we consider an estimation function which distributes density values in each cell to only movable areas in its neighboring cells. We do not assume any maps but exploit a density map to estimate movable areas in this function.

Fig. 5.2(c) and Fig. 5.4 show an example of an update by this limited diffuse estimation function and its pseudo-code, respectively. In this function, for each direction (*i.e.* up, bottom, left and right), we calculate the average density of cells to which the distance from the diffused cell $d_{x,y}$ is less than m cells. Then, if the result is more than TH_{move}, $d_{x,y}$ is divided by the number of directions which satisfy the condition and diffused to them. In the same way as the diffuse estimation function, α is regarded for aging. This procedure is iterated $\lfloor t * V_{max}/s \rfloor$ times.

In the case of the limited diffuse estimation function, the number of cells which satisfy the condition varies every time it updates a density map. Thus, it is complicated to derive Δt_i precisely. For this reason, we use the same rule with the diffuse estimation function to determine Δt_i.

Hybrid Estimation Function. Because a density map is propagated among nodes hop by hop, the freshness of information in farther areas is lower. This means that it is sometimes hard to estimate movable areas in

```
for(step=0; step<floor(t*Vmax/s); step++){
 D'_i=D_i;
 foreach(d_(x,y) in D_i){
  expand_num=1;
  sum=0;
  for(j=1; j<=m; j++) sum+=d_(x+j,y);
  avg=sum/m; right=false;
  if(avg >= TH_move){
   right=true; expand_num++;
  }
  sum=0;
  for(j=1; j<=m; j++) sum+=d_(x-j,y);
  avg=sum/m; left=false;
  if(avg >= TH_move){
   left=true; expand_num++;
  }
  sum=0;
  for(j=1; j<=m; j++) sum+=d_(x,y+j);
  avg=sum/m; down=false;
  if(avg >= TH_move){
   down=true; expand_num++;
  }
  sum=0;
  for(j=1; j<=m; j++) sum+=d_(x,y-j);
  avg=sum/m; up=false;
  if(avg >= TH_move){
    up=true; expand_num++;
  }
  if(right) d'_(x+1,y)=d_(x+1,y)+1/expand_num*d_(x,y)*alpha;
  if(left) d'_(x-1,y)=d_(x-1,y)+1/expand_num*d_(x,y)*alpha;
  if(down) d'_(x,y+1)=d_(x,y+1)+1/expand_num*d_(x,y)*alpha;
  if(up) d'_(x,y-1)=d_(x,y-1)+1/expand_num*d_(x,y)*alpha;
  d'_(x,y)=1/expand_num*d_(x,y)*alpha;
 }
 D_i=D'_i;
}
return D_i;
```

Figure 5.4: Limited Diffuse Estimation Function

farther regions based on the limited diffuse estimation function as we described before. Hence, we combine both the diffuse estimation function and the limited diffuse estimation function and propose the hybrid estimation function. In the hybrid estimation function, for the cells in the proximity of the current position, the limited diffuse estimation function is used and the diffuse estimation function is applied to distant areas.

We define the areas around the current position as the cells included in R_i, and use the limited diffuse estimation function for cells included in R_i and the diffuse estimation function for other cells. Δt_i is determined according to the same manner as the diffuse estimation function for the simplicity.

Recording Area of Presence

Each element $d'_{x,y}$ after recording node i's area of presence is calculated as defined below.

$$d'_{x,y} = \begin{cases} d_{x,y} + \dfrac{1}{n}, & \text{if } (x,y) \in R_i; \\ d_{x,y}, & \text{otherwise.} \end{cases} \tag{5.3}$$

where n denotes the number of elements in R_i. In this formula, the larger the size of R_i, the smaller the value added to each cell in R_i becomes.

Merging Density Maps

When a node i receives a density map D_j from another node j, node i merges D_i with D_j. Because each density map does not include any information which indicates the freshness of the density information in each cell, we regard a higher density as more fresh (*i.e.* newer) information. This policy is based on the observation that the density in each cell is diffused as time passes and hence a higher density is likely to be a fresher information. In merging of density maps, for each cell (x, y), the value $d'_{x,y}$ after the merging is computed as below.

$$d'_{x,y} = \max\{d^i_{x,y}, d^j_{x,y}\} \tag{5.4}$$

5.2.3 Getting Node Distribution from a Density Map

The node distribution is obtained from a density map D by finding cells with the highest density in a greedy fashion. The algorithm is described below in details.

1. Find a cell $c = (x_c, y_c)$ with the highest density d_{x_c,y_c}. If the densities of all cells are zero, terminate the distribution estimation.

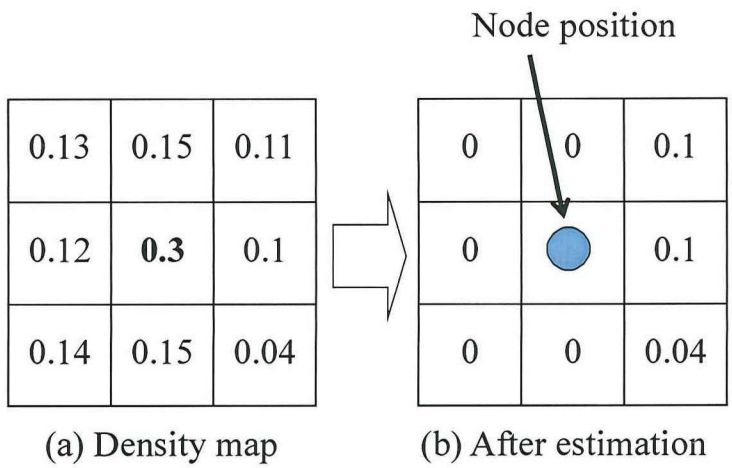


Figure 5.5: Estimation of Node Distribution from a Density Map

2. Set an estimated position of a node to a point in c and initialize a set C of cells to c.

3. If $d_{x_c,y_c} \geq 1.0$, subtract 1.0 from d_{x_c,y_c} and return to the first step.

4. Repeat adding neighboring cells of all cells in C to C until $\sum_{e \in C} d_{x_e,y_e} \geq 1.0$. Here, a neighboring cell of a cell c is a cell that shares any border or corner of c. If the algorithm cannot find any neighboring cells, terminate the distribution estimation.

5. Subtract 1.0 from C. For this purpose, we sort cells in C into descending order $c(1), c(2), \ldots, c(n)$ in terms of the density. For $c(1), c(2), \ldots, c(m-1), c(m), c(m+1), \ldots, c(n)$, set $d_{x_{c(1)},y_{c(1)}}, \ldots, d_{x_{c(m-1)},y_{c(m-1)}}$ to 0 where $\sum_{i=1}^{m-1} d_{x_{c(i)},y_{c(i)}} < 1.0$ and $\sum_{i=1}^{m} d_{x_{c(i)},y_{c(i)}} \geq 1.0$. Then, subtract $1.0 - \sum_{i=1}^{m-1} d_{x_{c(i)},y_{c(i)}}$ from $d_{x_{c(m)},y_{c(m)}}$, and go to the first step.

Intuitively, the algorithm sorts densities in descending order, and iterates the estimation of node positions from the cell with the highest density. At that time, we have to subtract 1.0 in total from the density map since a density of 1.0 corresponds to the presence of one node. Figure 5.5 shows an example of this algorithm. In Fig. 5.5(a), the highest density is 0.3 in the center cell, which is less than 1.0. Therefore, the densities

of the neighboring cells and the center cell are sorted in descending order $\{0.3, 0.15, 0.15, 0.14, 0.13, 0.12, 0.11, 0.1, 0.04\}$. Then, all values larger than 0.11 are set to 0 since $(0.3 + 0.15 + 0.15 + 0.14 + 0.13 + 0.12) = 0.99$ and $(0.3+0.15+0.15+0.14+0.13+0.12+0.11) = 1.1$. Finally, $1-0.99 = 0.01$ is subtracted from 0.11, and a node position is estimated by choosing a point in the center cell as shown in Fig. 5.5(b).

5.2.4 Reduction of Communication Overhead

Each node i sends its density map D_i every t seconds. The data size of D_i is inversely proportional to the size s^2 of a cell and proportional to the size of the target region. We introduce a technique which adjusts the view of a density map sent to neighbors, depending on the number of neighbors, in order to pursue the trade-off between the communication overhead and the accuracy.

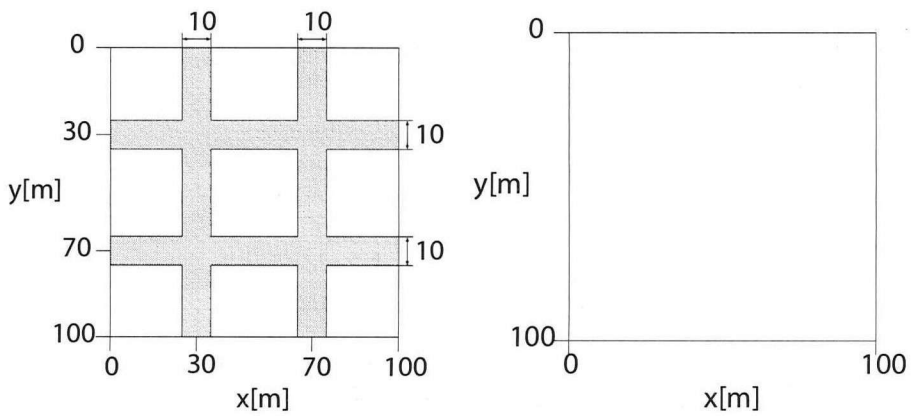
We denote a sub-density map of D_i as $\hat{D}_i$ hereafter. Ideally, it is better to send a density map D_i every t seconds in order to propagate the density information to distant areas for a higher accuracy. However, if the density around a node is high, it seems enough to send density maps from a few nodes in the surroundings because the information in distant areas is likely to be very similar among those density maps.

Based on this idea, our technique uses a sub-density map $\hat{D}_i$ of which the size is fixed and smaller than the density map. Every t seconds, each node i selects either its density map D_i or its sub-density map $\hat{D}_i$ to broadcast. The density map is selected with the probability of $1/N_i$ where N_i is the number of i's neighbor nodes. In addition, node i broadcasts D_i only if it has not sent D_i in the last T seconds in order to guarantee that a density map is sent in a certain period of time.

5.3 Experimental Results

5.3.1 Settings

We have evaluated the performance of the proposed method using a network simulator MobiREAL[52]. For simulation, we have used two maps of



(a) Manhattan (b) free-space

Figure 5.6: Simulation Maps

which the sizes are $100m \times 100m$. *Manhattan* in Fig. 5.6(a) which has 4 intersections and roads of 10m width, and *free-space* in Fig. 5.6(b). In the Manhattan map, nodes can only exist on roads, and in every map nodes were deployed uniformly before simulations. Nodes move along a road with a constant velocity which is randomly chosen from [0.1, 1.0](m/s). Each node changes its direction to the opposite if it encounters a border, and randomly chooses one of the three directions except the backward direction if it enters an intersection. In the free-space map, the random waypoint mobility model[53] with pause time 0 and the moving speed range of [0.1, 1.0] (m/s) was used. 200 nodes moved according to the above mobility models in each scenario. The length s of grid cells was set to 2m. We used the radio range of 10m and the network bandwidth of 1Mbps. We have assumed the location information R_i is obtained by GPS and given as a square region of size $49m \times 49m$ of which the center is the real node position. The simulation settings are summarized in Table 5.1. We have empirically decided threshold values shown in Table 1.

Through the analysis of simulation results, we confirmed that the accuracy of the estimated node distribution is very similar among the nodes of different initial locations and moving speeds. Therefore, in the following results, we focus on the density map of a particular node (this node is denoted as p hereafter) if no explicit explanation is given. We have measured the

Table 5.1: Simulation Settings

Parameters	Manhattan	free-space
estimation function $f(D)$	hybrid (m=10)	diffuse
available density threshold TH_d (node/cell)	0.01	0.015
density threshold ε for location registration interval (node/cell)	0.002	0.002
sub-density map Tx interval t(s)	2	2
density map Tx interval T(s)	10	10

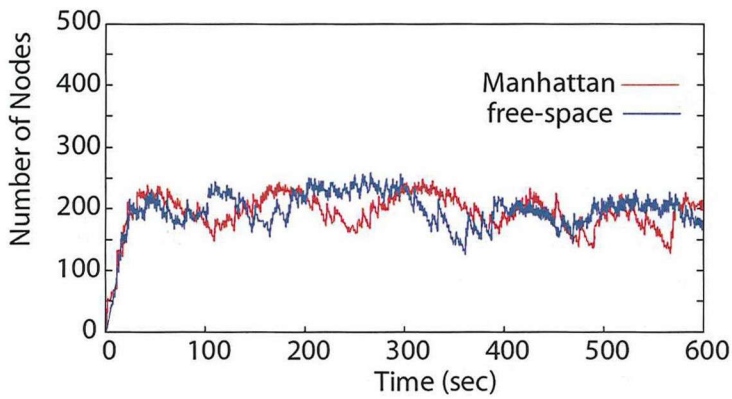


Figure 5.7: Time vs. Estimated Number of Nodes in a Density Map

positioning error to assess our method since it is quite intuitive and understandable not only for applications but also for people. By measuring the positioning error, we can compare our approach with the case where GPS positions of all the nodes are collected.

5.3.2 Results

Accuracy of a Number of Nodes

Fig. 5.7 shows the estimated number of nodes in the two maps, along the progress of the simulation time. Table 5.2 also shows the average number of nodes in each case. We can see that these averaged values are very close to the original values. In all the cases, large errors between the estimated

Table 5.2: Average Number of Nodes in an Estimated Density Map

	Estimated # of nodes
Manhattan (21s~600s)	199.788
free-space (21s~600s)	202.720

Table 5.3: Average Kendall's τ

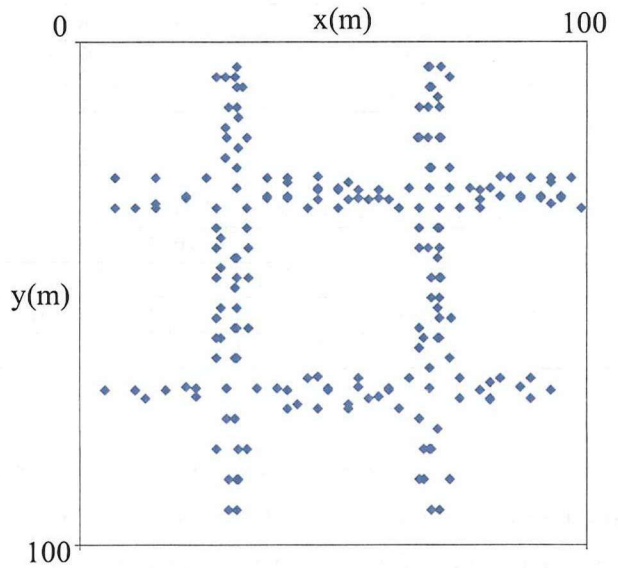
	Granularity			
	1×1	2×2	5×5	10×10
Manhattan (61s~600s)	0.680	0.718	0.735	0.761
free-space (61s~600s)	0.588	0.621	0.662	0.696

and real node densities were measured before 30sec. because it is the initial phase of the simulation where each node had started to collect information about the others and the density maps covering whole areas had not been constructed yet. Therefore, we focus on the state after 30sec., where the estimated number of nodes was stable with small errors from the real density.

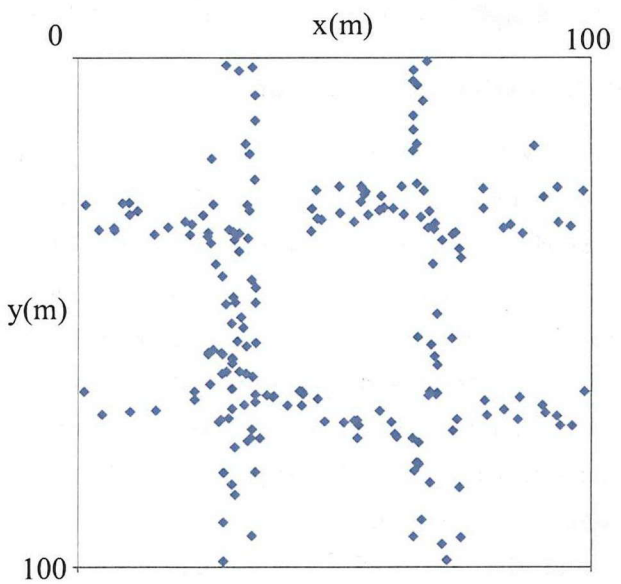
Similarity of the Estimated Distribution

To see the similarity between the estimated density distribution and the real density distribution, we used Kendall's τ[54]. Here, we introduce the concept of *granularity* to compare the two distributions. The granularity is represented by $g \times g$, which means that $g \times g$ cells are considered as one larger cell in computing the Kendall's τ. We have changed this granularity from 1×1 to 10×10. The results are shown in Table 5.3. The average Kendall's τ is increasing as the granularity becomes larger in most cases. This is natural because the values of density are often expanded to wider regions (*i.e.* the outside) by the diffuse estimation function.

Fig. 5.8(a) and Fig. 5.8(b) show the estimated node distribution of a node p at time 450sec. and its corresponding real node distribution in the case of the Manhattan map. By comparing the real node distribution with the estimated one, we can see some errors in each node position. However, we can also observe that the estimated node distribution has dense and sparse areas quite similar to the real ones. Lines of nodes imply the roads



(a) Estimated (Manhattan)



(b) Real (Manhattan)

Figure 5.8: Real Node Distribution and Estimated Node Distribution of Node p (at $450sec.$)

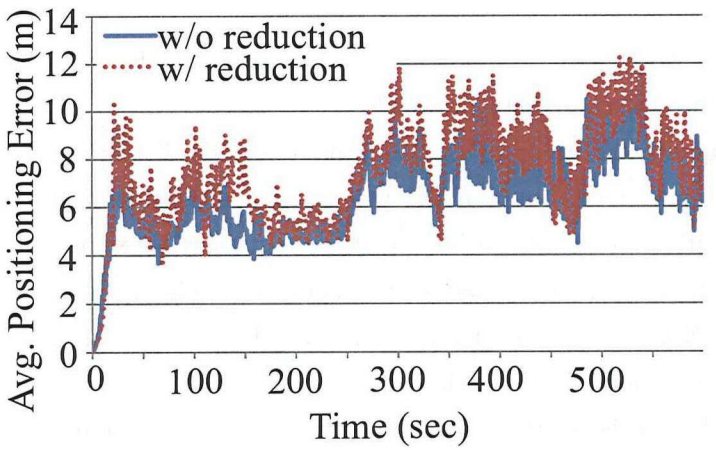
Table 5.4: Avg. Positioning Errors in Estimated Node Distribution of Node p

	Manhattan	free-space
w/o reduction	10.03	6.50
w/ reduction	10.89	7.39

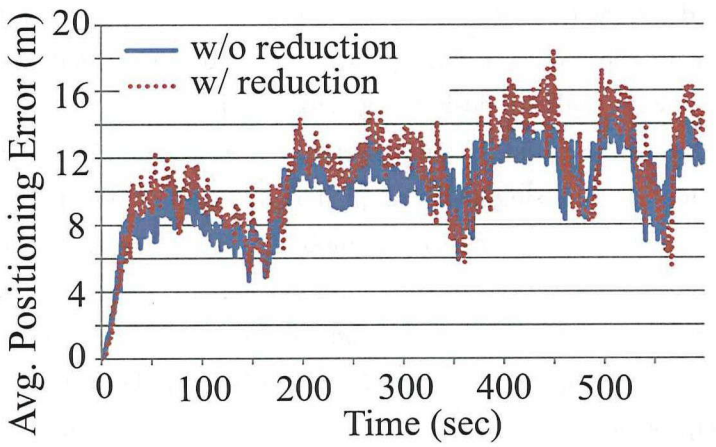
in the real world and this indicates the estimated node distribution well captures the real node distribution.

To evaluate the accuracy of the estimated node distribution, we focus on positioning errors between the estimated positions and the real positions. Positioning errors can be defined as distances between nodes in the estimated node distribution and its corresponding nodes in the real node distribution. Note that we never know the node identification in the estimated distribution. Therefore, for each estimated position, the nearest node in the real distribution is regarded as the corresponding node, and the distance between the estimated position and the real position of the corresponding node is used for calculating positioning errors. Here, each node in the real distribution is selected as a corresponding node of an estimated position only once so as to evaluate positioning errors properly.

Fig. 5.9(a) and Fig. 5.9(b) show averages of positioning errors (denoted as a solid line) in each simulation map. The averages of positioning errors in the both maps fluctuate due to the mobility. When the node p is in the proximity of the intersections or the center of the map, averages of positioning errors will be smaller because the node p can receive fresher information from different directions in such areas efficiently, and the accuracy of the density map of node p can be improved. Table 5.4 shows the average of positioning errors throughout the whole simulation. We can see that the average positioning error in the Manhattan map is about 10m and that in the free-space map is less than 10m approximately. These results mean that the proposed method can correctly estimate the node distribution in terms of the street level considering the road width of 10m and the road segment length of 40m in the Manhattan map. From the above results and observations of the average positioning error as a quantitative criterion, we can confirm the effectiveness of the proposed method.



(a) free-space



(b) Manhattan

Figure 5.9: Time vs. Average Positioning Errors in Estimated Distribution of Node p

Table 5.5: Comparison of Average Bandwidth per Node

	Manhattan	free-space
w/o reduction (kbps)	40	40
w/ reduction (kbps)	16.16	20

5.3.3 Reduction in Communication Overhead

In our technique, the target region is divided into cells. The number of cells is 2500 in the default simulation setting, and we assume that each cell requires 4 bytes. Then, the data size of a density map is 10 Kbytes. Each node sends its density map periodically and hence the communication overhead may be large. To reduce this communication overhead, we use a sub-density map as we mentioned in Sec. 5.2.4.

In order to see the effect of this scheme, we evaluated the amount of traffic. The result is shown in Table 5.5. We could confirm that our scheme could reduce approximately 50%-75% of the original traffic.

From the results shown in Fig. 5.9(a) and Fig. 5.9(b), we can see the average of positioning errors increases as time elapses in most cases with message reduction (denoted as dotted line), compared with the cases without message reduction. The second row of Table 5.4 also shows the averages of positioning errors with message reduction are 109% and 114% as much as those without message reduction in the Manhattan map and in the free-space map respectively. Obviously, there is a trade-off between the communication overhead and the accuracy of an estimated node distribution. Therefore, it is important to determine the parameters on the communication appropriately.

5.4 Discussion

The proposed method uses a cell matrix to represent a density map. The cell matrix facilitates the computation like merging and the mobility estimation, while the data size may be large, depending on both the region and cell sizes. In WSNs, there is a method to build a contour map of the data sensed by wireless sensor nodes [35, 55]. Some other possibilities use some encoding technique to compress the map. We are trying to clarify their advantages and disadvantages in terms of the trade-off between the computation overhead

and the data size.

We also discuss another important issue on the position information. In the proposed method, each node may provide its position information with some error range. This has the following two advantages, (i) robustness to position errors caused by the GPS or other measurements such as position estimation methods like Sextant[56] and UPL[41] due to their likelihood estimation in range-free localization, and (ii) privacy protection in which intentionally randomized positions obscure the true position.

The maximum speed V_{max} affects the estimation accuracy. If we use overestimated maximum speeds, the accuracy degrades because the estimated density spreads faster than the real speeds. In this sense, the maximum speed used in the simulation is overestimated since we have used 1.0m/s as the maximum speed although the real speeds uniformly distributed within [0.1, 1.0](m/s). Nevertheless, our approach has achieved a reasonable performance.

5.5 Conclusion

In this chapter, we have proposed a method for pedestrians to self-estimate the node distribution in their proximity in real-time using ad-hoc wireless communications among these nodes. We have conducted simulation experiments to see the accuracy and the communication overhead of the proposed method. Through quantitative evaluation by measuring positioning errors, we have confirmed the average position error is less than 10m, which is comparable with GPS errors. This result indicates our method estimates the node distribution accurately.

One of our potential application domain is personal navigation. In huge shopping centers and fireworks festivals (in the case of Japan) in which many people get around, observing their locations through their mobile terminals will be helpful not only for commercial use but also for safe navigation toward exits.

Assuming these potential application examples, we are planning to conduct simulations in more realistic environments, to determine appropriate parameter settings and to validate the usefulness of the method. Further-

more, the autonomy of the protocol is our important goal where protocol parameters like message transmission intervals can be autonomously converged into appropriate values in each cell depending on its neighborhood densities for zero-configuration.

Chapter 6

Conclusion

In this thesis, the following two research topics have been studied: (1) a service design technique which can reduce both of the network and server loads on an overlay network which is composed of repositories storing the sensig data and servers to process it, and (2) a pure-serverless service protocol which can be executed on a mobile ad-hoc network, and therefore in which there is no workload of servers essentially.

As the first topic of this thesis, we have proposed a method to derive execution sequences of a given application-level service that is executed by cooperative servers on overlay networks in Chapter 3. We also have designed and developed a service execution platform. The proposed method assumes that a service consists of service components, and it can derive optimal allocation of components that does not overload network links and servers. Using the platform, services can be installed and executed easily on real networks. The platform can measure network loads and CPU loads continuously, and can adaptively determine new allocation optimizing the system load in the environment. We have conducted experiments on PlanetLab to validate our method. We have confirmed that the proposed method could derive efficient execution sequences and they could achieve higher throughput than the other sequences.

As the second topic of this thesis, we have proposed a method for pedestrians to self-estimate the node distribution in their proximity in real-time using ad-hoc wireless communications among these nodes in Chapter 5. We have conducted simulation experiments to see the accuracy and the commu-

nication overhead of the proposed method. Through quantitative evaluation by measuring positioning errors, we have confirmed the average position error is less than 10m, which is comparable with GPS errors. This result indicates our method estimates the node distribution accurately.

One of our potential application domain is personal navigation. In huge shopping centers and fireworks festivals (in the case of Japan) in which many people get around, observing their locations through their mobile terminals will be helpful not only for commercial use but also for safe navigation toward exits.

Assuming these potential application examples, we are planning to conduct simulations in more realistic environments, to determine appropriate parameter settings and to validate the usefulness of the method. Furthermore, the autonomy of the protocol is our important goal where protocol parameters like message transmission intervals can be autonomously converged into appropriate values in each cell depending on its neighborhood densities for zero-configuration.

As future work, we leave secure methods to avoid sending raw sensing data to remote devices or servers since sensing data is a kind of personal information. It is desired that data aggregation among neighbor devices should be considered to ensure security, satisfying constraints for workloads of network and servers.

Acknowledgement

First of all, I would like to gratefully acknowledge enthusiastic supervision of Professor Teruo Higashino during this research. I also would like to express my deepest appreciation for his great support and generous encouragement in my student life.

I am very grateful to Professor Hirotaka Nakano, Professor Koso Murakami and Professor Masayuki Murata of Osaka University for their invaluable comments and helpful suggestions concerning this thesis.

I would like to express my heartfelt gratitude to Associate Professor Hirozumi Yamaguchi for inestimable advices and innumerably valuable comments.

My enormous thankfulness goes to Assistant Professor Akihito Hiromori and Assistant Professor Akira Uchiyama whose comments and advices are invaluable for my study.

Assistant Professor Takaaki Umedu's technical supports and precious advices for experiments are gratefully acknowledged.

Finally, let me thank to everyone of Higashino Laboratory for their feedback, encouragement and support for my research.

Bibliography

[1] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use Volume 1: Basic Concepts.* Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 1997.

[2] Jens Wepper and Paul Lukowicz. Collaborative crowd density estimation with mobile phones. In *Proc. of International Workshop on Sensing Applications on Mobile Phones (PhoneSense)*, 2011.

[3] T. Horanont and R. Shibasaki. An implementation of mobile sensing for large-scale urban monitoring. In *Proc. of the International Workshop on Urban, Community, and Social Applications of Networked Sensing Systems (UrbanSense)*, 2008.

[4] A.V. Ratzer, L. Wells, H.M. Lassen, M. Laursen, J.F. Qvortrup, M.S. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN Tools for editing, simulating, and analyzing coloured Petri nets. In *Proc. of 24th Int. Conf. on Application and Theory of Petri Nets*, 2003.

[5] D. Xu and K. Nahrstedt. Finding service paths in an overlay media service proxy network. In *Proc. of Int. Conf. on Multimedia Computing and Networking 2002 (MMCN2002)*, 2002.

[6] M. Wang, B. Li, and Z. Li. sFlow: Towards resource-efficient and agile service federation in service overlay networks. In *Proc. of 24th Int. Conf. on Distributed Computing Systems (ICDCS2004)*, 2004.

[7] X. Gu, K. Nahrstedt, and B. Yu. Spidernet: An integrated peer-to-peer service composition framework. In *Proc. of IEEE Int. Symposium on High-Performance Distributed Computing (HPDC-13)*, 2004.

[8] Baochun Li, Jiang Guo, and Mea Wang. iOverlay: A lightweight middleware infrastructure for overlay application implementations. In *Proceedings of the Fifth ACM/IFIP/USENIX International Middleware Conference (Middleware 2004), also Lecture Notes in Computer Science*, pp. 135–154, 2004.

[9] Didier Benza, Michel Cosnard, Luigi Liquori, and Marc Vesin. Arigatoni: A simple programmable overlay network. In *JVA '06: Proceedings of the IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing*, pp. 82–91. IEEE Computer Society, 2006.

[10] Boon T. Loo, Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Timothy Roscoe, and Ion Stoica. Implementing declarative overlays. *SIGOPS Oper. Syst. Rev.*, Vol. 39, No. 5, pp. 75–90, December 2005.

[11] Kazuyuki Shudo, Yoshio Tanaka, and Satoshi Sekiguchi. Overlay weaver: An overlay construction toolkit. *Computer Communications*, Vol. 31, No. 2, pp. 402–412, February 2008.

[12] Adolfo Rodriguez, Charles Killian, Sooraj Bhat, Dejan Kostic, and Amin Vahdat. MACEDON: Methodology for automatically creating, evaluating, and designing overlay networks. In *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI2004)*, pp. 267–280, 2004.

[13] H. Yamaguchi, K. El-Fakih, A. Hiromori, and T. Higashino. A formal approach to design optimized multimedia service overlay. In *Proc. of 15th ACM Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2005)*, pp. 57–62, 2005.

[14] OKI. Oki press release. `http://www.oki.com/en/press/2009/01/z08113e.html`.

[15] Andrew T. Campbell, Shane B. Eisenman, Nicholas D. Lane, Emiliano Miluzzo, Ronald A. Peterson, Hong Lu, Xiao Zheng, Mirco Musolesi, Kristóf Fodor, and Gahng-Seop Ahn. The rise of people-centric sensing. *IEEE Internet Computing*, Vol. 12, No. 4, pp. 12–21, 2008.

[16] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky. Seti@home-massively distributed computing for seti. *Computing in Science & Engineering*, Vol. 3, No. 1, pp. 78–83, 2001.

[17] Stefan M. Larson, Christopher D. Snow, Michael R. Shirts, and Vijay S. Pande. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2002.

[18] Kenji Saito and Kendai Miyazawa. Rapid p2p overlay network programming on a distributed reduction machine. In *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, CCNC'09, pp. 1262–1266, Piscataway, NJ, USA, 2009. IEEE Press.

[19] I. Baumgart, B. Heep, and S. Krause. Oversim: A scalable and flexible overlay framework for simulation and real network applications. In *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, pp. 87 –88, sept. 2009.

[20] Jordi Pujol-Ahulló, Pedro García-López, Marc Sànchez-Artigas, and Marcel Arrufat-Arias. An extensible simulation tool for overlay networks and services. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pp. 2072–2076, New York, NY, USA, 2009. ACM.

[21] Ludger Bischofs and Wilhelm Hasselbring. Analyzing and implementing peer-to-peer systems with the peerse experiment environment. *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on*, Vol. 0, pp. 311–315, 2009.

[22] Mansoor Alicherry and T.V. Lakshman. Network aware resource allocation in distributed clouds. In *Proceedings of the 31st IEEE International Conference on Computer Communications*, pp. 53–58. IEEE Computer Society, 2012.

[23] Hendrik Moens, Jeroen Famaey, Steven Latre, Bart Dhoedt, and Filip De Turck. Design and evaluation of a hierarchical application placement algorithm in large scale clouds. In *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management(IM)*, pp. 137–144. IEEE Computer Society, 2011.

[24] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Proceedings of the 29th conference on Information communications (INFOCOM)*, pp. 1154–1162. ACM, 2010.

[25] Fetahi Wuhib, Rolf Stadler, and Mike Spreitzer. Gossip-based resource management for cloud environments. In *Conference on the 6th IEEE International Conference on Network and Service Management (CNSM)*, pp. 1–8. IEEE Computer Society, 2010.

[26] Dejan Kovachev, Tian Yu, and Ralf Klamma. Adaptive computation offloading from mobile devices into the cloud. In *Proceedings of the 10th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pp. 784–791. IEEE Computer Society, 2012.

[27] Sougata Sen, Archan Misra, Rajesh Balan, and Lipyeow Lim. The case for cloud-enabled mobile sensing services. In *Proceedings of the first edition of the MCC Workshop on Mobile Cloud Computing*, pp. 53–58. ACM, 2012.

[28] Eugene Marinelli. Hyrax: cloud computing on mobile devices using mapreduce. In *Master thesis.* Carnegie Mellon University, 2009.

[29] Emiliano Miluzzo, Ramon Caceres, and Yih-Farn Chen. Vision: mclouds - computing on clouds of mobile devices. In *Proceedings of the 3rd ACM workshop on Mobile cloud computing and services(MCS)*, pp. 9–14. ACM, 2012.

[30] Fan Yang, Zhengping Qian, Xiuwei Chen, Ivan Beschastnikh, Li Zhuang, Lidong Zhou, and Guobin She. Sonora: A platform for

continuous mobile-cloud computing. In *Technical Report.* Microsof Research Asia, 2012.

[31] G. Korkmaz, E. Ekici, F. Özgüner, and Ü. Özgüner. Urban multi-hop broadcast protocol for inter-vehicle communication systems. In *Proc. of ACM International Workshop on VehiculAr Inter-NETworking, Systems, and Applications (VANET)*, pp. 76–85, 2004.

[32] C. Lochert, B. Scheuermann, and M. Mauve. Probabilistic aggregation for data dissemination in VANETs. In *Proc. of ACM International Workshop on VehiculAr Inter-NETworking, Systems, and Applications (VANET)*, pp. 1–8, 2007.

[33] B. Yu, J. Gong, and C. Xu. Catch-up: a data aggregation scheme for VANETs. In *Proc. of ACM International Workshop on VehiculAr Inter-NETworking, Systems, and Applications (VANET)*, pp. 49–57, 2008.

[34] J. Zhao and G. Cao. Vadd: Vehicle-assisted data delivery in vehicular ad hoc networks. *IEEE Transactions on Vehicular Technology*, Vol. 57, No. 3, pp. 1910–1922, 2008.

[35] I. Gupta, R.V. Renesse, and K.P. Birman. Scalable fault-tolerant aggregation in large process groups. In *Proc. of IEEE International Conference on Dependable Systems and Networks (DSN)*, pp. 433–442, 2001.

[36] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Operating Systems Review*, Vol. 36, No. SI, pp. 131–146, 2002.

[37] A. Boulis, S. Ganeriwal, and M.B. Srivastava. Aggregation in sensor networks: an energy accuracy trade-off. *Ad Hoc Networks*, Vol. 1, No. 2-3, pp. 317–331, 2003.

[38] M. Papadopouli and H. Schulzrinne. Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In *Proc. of ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pp. 117–127, 2001.

[39] D.K. Goldenberg, P. Bihler, M. Cao, J. Fang, B.D.O. Anderson, A.S. Morse, and Y.R. Yang. Localization in sparse networks using sweeps. In *Proc. of ACM International Conference on Mobile Computing and Networking (MobiCom)*, pp. 110–121, 2006.

[40] M. Li and Y. Liu. Rendered path: Range-free localization in anisotropic sensor networks with holes. In *Proc. of ACM International Conference on Mobile Computing and Networking (MobiCom)*, pp. 51–62, 2007.

[41] A. Uchiyama, S. Fujii, K. Maeda, T. Umedu, H. Yamaguchi, and T. Higashino. Ad-hoc localization in urban district. In *Proc. of IEEE International Conference on Computer Communications (Infocom)*, pp. 2306–2310, 2007.

[42] T. He, C. Huang, B.M. Blum, J.A. Stankovic, and T. Abdelzaher. Range-free localization schemes for large scale sensor networks. In *Proc. of ACM International Conference on Mobile Computing and Networking (MobiCom)*, pp. 81–95, 2003.

[43] Y. Sekimoto, R. Shibasaki, H. Kanasugi, T. Usui, and Y. Shimazaki. Pflow: Reconstructing people flow recycling large-scale social survey data. *IEEE Pervasive Computing*, Vol. 10, No. 4, pp. 27–35, 2011.

[44] T. Horanont and R. Shibasaki. Nowcast of urban population distribution using mobile phone call detail records and person trip data. In *Proc. of International Conference on Computers in Urban Planning and Urban Management (CUPUM)*, 2011.

[45] NTT DOCOMO Technical Journal Editorial Office. Measures for recovery from the great east japan earthquake using ntt docomo r&d technology. *NTT DOCOMO Technical Journal*, Vol. 13, No. 4, pp. 96–106, 2012.

[46] L. A. Cherkasova, V. E. Kotov, and T. Rokicki. On net modeling of industrial size concurrent systems. In *Proc. of 14th Int. Conf. on Application and Theory of Petri Nets 1993 (LNCS 691)*, pp. 552–561. Springer-Verlag, 1993.

[47] P. Huber and V.O. Pinci. A formal executable specification of the ISDN basic rate interface. In *Proc. of 12th Int. Conf. on Application and Theory of Petri Nets*, pp. 1–21, 1991.

[48] J.C.A. de Figueiredo and L.M. Kristensen. Using coloured Petri nets to investigate behavioural and performance issues of TCP protocols. In *Proc. of 2nd Workshop on Practical Use of Coloured Petri Nets and Design/CPN*, pp. 21–40, 1999.

[49] J. L. Rasmussen and M. Singh. Designing a security system by means of coloured Petri nets. In *Proc. of 17th Int. Conf. on Application and Theory of Petri Nets (LNCS 1091)*, pp. 400–419, 1996.

[50] H. Yamaguchi, K. El-Fakih, G. v. Bochmann, and T. Higashino. Deriving protocol specifications from service specifications written as predicate/transition-nets. *Computer Networks Journal*, Vol. 51, No. 1, pp. 2581–284, January 2007.

[51] Planetlab: An open plat form for developong, deploying, and accessing planetary-scale services. `http://www.planet-lab.org`.

[52] MobiREAL. Mobireal simulator. `http://www.mobireal.net/`.

[53] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. of ACM International Conference on Mobile Computing and Networking (MobiCom)*, pp. 85–97, 1998.

[54] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical recipes in C: the art of scientific computing.* Cambridge Univ. Press, second edition, 1992.

[55] Y. Xu, W.-C. Lee, and G. Mitchell. CME: a contour mapping engine in wireless sensor networks. In *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, pp. 133–140, 2008.

[56] S. Guha, R. Murty, and E. Sirer. Sextant: a unified node and event localization framework using non-convex constraints. In *Proc. of In-*

ternational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), pp. 205–216, 2005.