



THE HONG KONG
POLYTECHNIC UNIVERSITY

香港理工大學

Pao Yue-kong Library
包玉剛圖書館

Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

By reading and using the thesis, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.
2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.



THE HONG KONG
POLYTECHNIC UNIVERSITY
香港理工大學

Department of Computing

Thesis for Master of Philosophy

**A Measurement-Based Approach to Support Real-time Object-Based
Applications Timeliness**

Student name Lo Chun Tat

Student ID 0190

4th Jan 2006

A thesis submitted in partial fulfillment of requirements for the Degree of Master of
Philosophy.

1



Pao Yue-kong Library
PolyU · Hong Kong

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written nor material which has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signature)

Lo Chun Tat (Name of Student)

ABSTRACT

In this paper the novel, original transfer policy framework, namely, the *statistical distribution independent transfer policy model* (SDITPM) for making sound migration decision in a real-time-manner, is proposed. This framework, which can be incorporated as part of a logical entity (e.g. a logical server) construct, steers the latter effectively in the migration process over a sizeable network such as the Internet. As a result it relieves network congestion and shortens round-trip time through load balancing, which comes naturally from the guided object mobility. The SDITPM framework is a significant contribution to time-critical applications over the Internet because of the shortened service roundtrip time (RTT) in the client/server interactions. Applications for the Internet are intrinsically object-based and exploit the potential speedup supported by the distributed hardware in the underlying network. The objects interact in an end-to-end many-clients-to-one-server relationship called the asymmetric rendezvous via logical channels. If the service RTT for every client/server interaction is shortened, then the overall execution time of the application will be reduced. Therefore, the SDITPM is immensely useful in helping the harnessing of service RTT and making the Internet more usable for real-time computing.

The SDITPM mechanism leverages system metrics on the fly, independent of their distribution types (e.g. long-range dependence (LRD) and short-range dependence (SRD)). That is, the SDITPM framework works by direct data measurement statistically, supported by the Convergence Algorithm (CA), which is

derived from the Central Limit Theorem. Every system metric being leveraged is called a primary metric, from which the secondary metrics are derived. The present SDITPM framework is basically a PID or “P+I+D” controller (P for proportional control, I for integral control and D for derivative control). The P and D control elements are secondary metrics and the I control sums their control effects for the specific r^{th} control region. In the SDITPM context every primary metric represents a control plane of four regions. Three of them are dedicated separately to PID, PI and DI controls, and the fourth one is inert.

From every leveraged primary metric the SDITPM mechanism derives two secondary ones. For example, if the metric is the logical server’s queue length Q , then for the r^{th} control region the following are true, a) the P control is the change in Q between two time points (e.g. t_1 and t_2) and b) the D control is the corresponding rate of change, $\frac{dQ}{dt} = \frac{(Q_{t_2} - Q_{t_1})}{(t_2 - t_1)}$. The control plane/matrix of the primary metric is actually defined by two *objective functions*, $\{0_P, \Delta_P\}^2$ for the P control (rows in matrix) and $\{0_D, \Delta_D\}^2$ for the D control (columns in the same matrix). The four control regions defined by these two objective functions are: a) Region 1 (i.e. r^1 or r^{PID}) defined by the $[P+, D+]$ pair of positive values for PID control, b) Region 2 (i.e. r^2 or r^{DI}) defined by $[P-, D+]$ for “D+I” or DI control, c) Region 3 (i.e. r^3 or r^{PI}) defined by $[P+, D-]$ for PI control, and Region 4 (i.e. r^4 or r^{Inert}) defined by $[P-, D-]$ as an inert or “don’t care” state. The regional control (i.e. r^{th}) only takes the positive value(s) of P and D (i.e. P+ and D+) into account. When the effects of the D and P control are added for the regional control the summation is the

integral control effect. In the SDITPM context summation is called the planar integration if only a single metric is being leveraged. If n metrics are leveraged simultaneously, the current control effect for a region r_n^x is $C_{r_n^x} = \sum_{i=1}^n (P_i^x, D_i^x)$, where x indicates the control region (e.g. r^1 or r^{PID}). (P_i^x, D_i^x) represents the P and D deviation errors for the objective functions: $\{0_P, \Delta_P\}^2$ and $\{0_D, \Delta_D\}^2$ respectively. The deviation error ψ is the difference between the sampled value S and the given safety margin Δ (e.g. Δ for P control is Δ_P) conceptually; $\psi = |S - \Delta|$ implies $\psi_P = |S_P - \Delta_P|$ and $\psi_D = |S_D - \Delta_D|$. The emphasis is that the SDITPM framework should leverage only the following conditions: a) positive S and b) $S > \Delta$ for formulating its P, D and I control elements. The migration decision depends on the dominant region, which should currently have the highest $C_{r_n^x}$ value. Which control region is dominant is totally decided by the system dynamics. $C_{r_n^x}$ is called the regional transfer probability TP_r (i.e. $TP_r = C_{r_n^x}$), and the threshold for the region is Th_r . The dominant TP_r becomes the overall transfer probability TP_o and Th_r of the same region is the overall Th_o threshold. A migration decision at the transfer policy level is affirmed for the condition $TP_o > Th_o$. Whether the migration decision will be realized or not depends on the availability of a suitable target node for the logical server to migrate to. Finding the target node is the decision of the location policy, which is not within the scope of the present research. In the SDITPM verification exercise, however, the effect of the location policy is produced

by using the A^4 visualization package, which is also part of the original contribution by this thesis.

ACKNOWLEDGEMENTS

I would like to thank Dr. Allan Wong who has been guiding me fervently in my MPhil thesis. I am grateful to Dr. Wilfred Lin, Dr. Richard Wu and Miss May Ip for giving me suggestions in resolving different technical problems. Last but not the least I thank the Department of Computing for given me partial financial support.

TABLE OF CONTENT

CERTIFICATE OF ORIGINALITY	2
ABSTRACT	3
ACKNOWLEDGEMENTS	7
TABLE OF CONTENT	8
LIST OF FIGURES	10
LIST OF TABLES	13
CHAPTER 1. INTRODUCTION	14
1.1 From the Asymmetric Rendezvous Perspective	18
1.2 Problem Statement and Objectives	22
1.3 Choice of Research Methodology	27
1.4 Outline of the Thesis	29
CHAPTER 2. BACKGROUND AND RELATED WORK	33
2.1 The Convergence Algorithm.....	38
2.2 From Process Migration to Object Migration	40
2.3 Load Balancing in a Decentralized Computing Environment	44
2.4 Recap of Effective Distributed Processing	48
2.5 Connective Summary	53
CHAPTER 3. THE NOVEL SDITPM FRAMEWORK	55
3.1 The Conceptual Framework.....	55
3.2 The Concept of PID Control by SDITPM.....	58
3.3 Connective Summary	68

CHAPTER 4. VERIFICATION EXPERIMENTS	70
4.1 Static Verifications.....	75
4.1.1 Single Metric.....	75
4.1.1.1 Memory Utilization.....	75
4.1.1.2 CPU Utilization.....	79
4.1.1.3 Context Switching Time	80
4.1.2 Leveraging Two Metrics.....	81
4.1.3 Leveraging Three Metrics.....	82
4.2 Dynamic Metric Leveraging	84
4.3 Connective Summary	84
CHAPTER 5. DYNAMIC SDITPM VERIFICATION AND VISUALIZATION OF OBJECT MOBILITY	86
5.1 Some Adapted A^3 Framework Details.....	89
5.2 A Walkthrough of the Original A^3 Framework	91
5.2.1 Membership Registration.....	93
5.2.2 Tracing the Object Mobility.....	98
5.3 Tracing SDITPM Object Mobility.....	101
5.4 The Distinctive A^4 Features	103
5.4.1 Leveraging Metrics and Their Statistics	108
5.5 Timing Analysis of the SDITPM Prototype	111
5.6 Connective Summary	112
CHAPTER 6. CONCLUSION, FUTURE WORK AND ACHIEVEMENTS	113
BIBLIOGRAPHY	122

LIST OF FIGURES

Figure 1.1. Master/slave object-based programming model.....	15
Figure 1.2. Client/server asymmetric rendezvous over an Internet TCP channel.	18
Figure 1.3. A real-life heavy-tailed RTT distribution over 24 hours	20
Figure 1.4. The RTT of a logical channel	25
Figure 1.5. The Iterative Research Methodology (IRM)	29
Figure 2.1. M_i measured by MCA for the TCP channel between Hong Kong PolyU and LaTrobe University in Australia.....	40
Figure 2.2. Evolution of logical entity migration.....	43
Figure 2.3. Concept of a repository for nodes' statuses.....	46
Figure 3.1. The four control regions of a primary metric	59
Figure 3.2. Leveraging multiple primary metrics by SDITPM.....	61
Figure 3.3. VTune timing analysis of the Java MCA implementation (i.e. M^3RT)	66
Figure 3.4. Simple linear regression of 13 queue length samples.....	67
Figure 4.1. The Aglets environment for the SDITPM verification experiments ..	71
Figure 4.2. The memory utilization profile/trace	76
Figure 4.3. SDITPM migration decisions for the memory utilization in Figure 4.2	78
Figure 4.4. A CPU utilization profile.....	80
Figure 4.5. Migration decisions by SDITPM based on a CPU utilization trace ...	80
Figure 4.6. Context switching time (CST) profile/trace	81

Figure 4.7. Migration decisions for the CST trace in Figure 4.6.	81
Figure 4.8. Two superimposed metrics, memory and CPU utilizations	82
Figure 4.9. Migrations by leveraging memory and CPU utilizations simultaneously	82
Figure 4.10. Superimposed context switching time, memory and CPU utilizations	83
Figure 4.11. Migrations by leveraging metrics shown in Figure 4.10 (threshold = 50)	83
Figure 4.12. Migrations by leveraging the metrics in Figure 4.1 (threshold = 180)	84
Figure 5.1. Three villages interact through the mediators to form a Public Intranet (PI)	89
Figure 5.2. CI for the coordinator - atp://u5x-170:8001/	92
Figure 5.3. Member User Interface (MUI) at member node atp://u5x-172:8001/	93
Figure 5.4 MUI for node (atp://u5x-172:8001/) shows the connection between atp://u5x-170 and atp://u5x-172 after a successful registration operation by atp://u5x-172	95
Figure 5.5. CI the atp://u5x-170) coordinator lists the current member	95
Figure 5.6. The mediator atp://u5x-172:8001/ has joined two villages	96
Figure 5.7. Node atp://u1x-139:900/ is also a member in the village managed by coordinator atp://ulx-137:9100/ as shown by its CI.....	97
Figure 5.8. Node atp://u5x-172 has de-registered from atp://u5x-171.....	98
Figure 5.9. Object placed/dispatched to node atp://b5x-178.....	99

Figure 5.10. Object automatically migrated to node atp://u5x-172	100
Figure 5.11. Resident evaluators and a coordinator's TP_o table	102
Figure 5.12. A migration path by A^3 framework without SDITPM support	103
Figure 5.13. Migration path of a logical server supported by A^4 framework	103
Figure 5.14 Display of the SDITPM control values by the b5x-180 console's MUI	104
Figure 5.15. An example of collecting statistics after the logical server is dispatched	105
Figure 5.16 A L/T screen example.....	106
Figure 5.17. An example of when a migration decision was eventually materialized (from u5x-176 to u5x-178)	107
Figure 5.18. Consumer.class injects perturbations and sample remote statistics	108
Figure 5.19. Intrinsic regional control values of captured in atp://u5x-176:8001/	108
Figure 5.20. "Self" regional control values for the migration logged (from u5x-175 to b5x-178).....	109
Figure. 5.21 Statistics at time T1 when object migrated from u5x-176 to b5x-178 (refer to the L window in Figure 5.17).....	110
Figure 5.22 Comparing relevant TP_o values (refer to the PI in Figure 5.15)	110
Figure 5.23. One example of timing analysis of the SDITPM execution	111

LIST OF TABLES

Table 2.1 Some issues/problems that can be resolved by process / object mobility	37
Table 3.1. Three primary metrics (STK=3) and for computing $TP_{l,r}$; $TP_{rS}^c = TP_{l,r}$	65
Table 4.1. SDITPM decision matrix for a control plane (one metric)	74
Table 4.2. Real-time calculations of M_i/M_{i-1} and dQ/dt	77
Table 5.1. Comparison of some Agent Systems	88

CHAPTER 1. INTRODUCTION

In this thesis the novel *statistical distribution independent transfer policy model* (*SDITPM*), which helps a logical server make sound transfer policy decisions, is proposed. These decisions are important because they guide the servers to migrate/relocate from overloaded hosts to less congested ones. The server mobility in the process has naturally created a load balancing effect that shortens the average service roundtrip time (RTT) for the server [Shirazi95]. Therefore, the contribution by SDITPM to time-critical applications over the Internet is significant. To reap the benefits from server mobility the SDITPM implementation should be included as a component of a logical server construct. Normally object migration should involve two policies [Coulouris01]. The first is the transfer policy that decides if a migration should occur. The second is the location policy that finds a suitable remote node/host as the migration destination.

Applications running on the Internet are object-based, and the cognate objects interact in a client/server relationship over logical TCP (Transmission Control Protocol) channels [RFC793]. Service RTT is the interval between the point when a client has made a service request and the moment when it has received the result correctly. If the host of a logical server is congested, the average service RTT would be prolonged by delays of various origins, including queuing, context switching, and retransmissions. If a logical server can migrate from its congested host to a less busy node, the service RTT is automatically reduced due to the effect of load balancing, supported by object mobility. The SDITPM guides the object

migration process by leveraging *primary metrics* (e.g. server's queue length). From every primary metric the corresponding *secondary* metrics are derived for the sake of computing the *overall transfer probability* TP_o . Conceptually the migration decision by the transfer policy is affirmative provided that the following condition is satisfied: $TP_o > Th_o$, where Th_o is a predefined threshold for effecting an object migration decision.

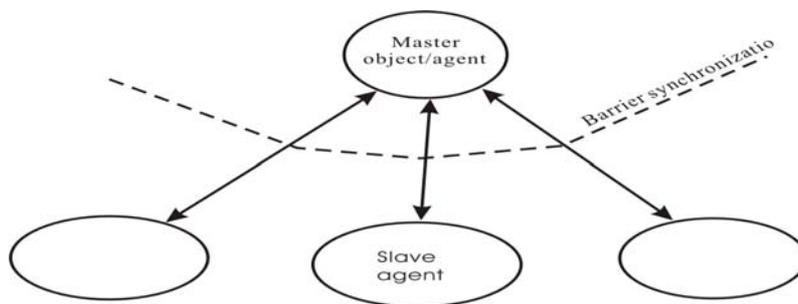


Figure 1.1. Master/slave object-based programming model

Applications running on the Internet are usually object-based, and the cognate objects/agents interact in a client/server relationship by message passing [Lewandowski98]. For example, in Figure 1.1 a master object collaborates with the slave objects through barrier synchronization. The service roundtrip time (RTT) between the master (the client) and a slave (server) depends on the current loading conditions of the slave's host. Object-based computing provides computation speedup because of its intrinsic parallelism. The terms agent and object are interchangeably and liberally used in this paper. An agent is a logical entity/object that operates independently, and the type of service provided by an agent characterizes its "*attitude*" [Bradshaw97]. The novel *statistical distribution independent transfer policy model (SDITPM)* proposed in this paper supports object-

based computing over a sizeable network such as the Internet. It shortens the client/server RTT over a logical channel by helping the server make sound migration decisions at the transfer policy level. The benefit of a successfully guided object migration process is effective load balancing, which naturally shortens the service RTT.

Object mobility provides applications that run on the Internet with different advantages. For example, it allows a logical object/agent to move out of a “*hot-spot*” location to a less busy one. The resultant load-balancing effect naturally shortens the service RTT [Dillon00]. Object mobility is the backbone of the new *Remote Programming Paradigm* (RPP) [Cockayne97]. From the RPP angle a client can create and send agent probes to remote locations where there are large volumes of data. These probes process the data locally and send back only the results after they have finished the delegated tasks. This approach reduces the amount of interim communication traffic between the client and the server. The speedup potential and other advantages (e.g. program security and safety) by RPP have inspired the development of different mobile agent platforms. One of them is the Aglets by IBM [AGLETS], which is chosen for the SDITPM experiments because: a) it is stable, b) it is rich in user experience, and c) it is specifically designed for Internet applications and this makes the experimental results scalable and repeatable over the open Internet. The RPP approach is useful for e-commerce applications [Wong00], which include: a) searching a publication from e-book stores in the cyberspace (e.g. Amazon) and b) finding the cheapest air ticket over the web. Although the results from different researches (e.g. [Dillon00]) indicate that agent mobility indeed yields

high system performance, speedup of an application execution depends on the “degree of overlapped parallelism β [M^3RT]”.

If a distributed application is made up of N cognate mobile agents, the speedup depends on the asynchrony indicated by β among the object collaborations. The β effect depends on the programming model, which can be master/slave (M/S), iterative stages, process network or a combination of all these [Yeung98a]. For example, in every barrier synchronization (BS) cycle in Figure 1.1, the master tries to synchronize the slaves by sending them the same go-ahead signal. The slaves, however, may receive the signal at different times due to communication delay. Similarly, the slave agents may not complete their BS cycles at the same time because of the different loading conditions in their hosts. Yet, the master must receive all the partial results from slaves before it starts the next BS cycle. If the M/S program needs P number of BS passes to complete program execution with mean BS cycle time T_{BS} , the overall program execution time involving N agents is $T_{PE} = T_{BS} * P * N^{e^{-\beta}}$. If a M/S collaboration has little asynchrony, the β value would be large (i.e. highly parallel) implying $T_{PE} \approx T_{BS} * P * N^0 = T_{BS} * P$. This T_{PE} value with N^0 is typical for intrinsic SIMD (*single instruction multiple data*) multiprocessor architectures wherein all N processors are in full synchrony.

The load balancing effect by object migration can shorten T_{BS} , which is the service RTT between the client (master) and the slave (server), and thus T_{PE} . The novel SDITPM framework proposed in this paper helps Internet based distributed computing achieve the effect of $T_{PE} \approx T_{BS} * P * N^0$ effect.

1.1 From the Asymmetric Rendezvous Perspective

The client/server inter-object interaction is an *asymmetric rendezvous* (i.e. one server to many clients). The streams of service requests from different clients merge at the server's SAP (*service access point*) before entering the queue. For a popular server this merged traffic (indicated by "+" in Figure 1.2) could surge due to high incoming request rate. This can cause the queue to overflow its buffer easily. The consequence is widespread timeouts and retransmissions by the clients and ensuing network congestions [Lakes97].

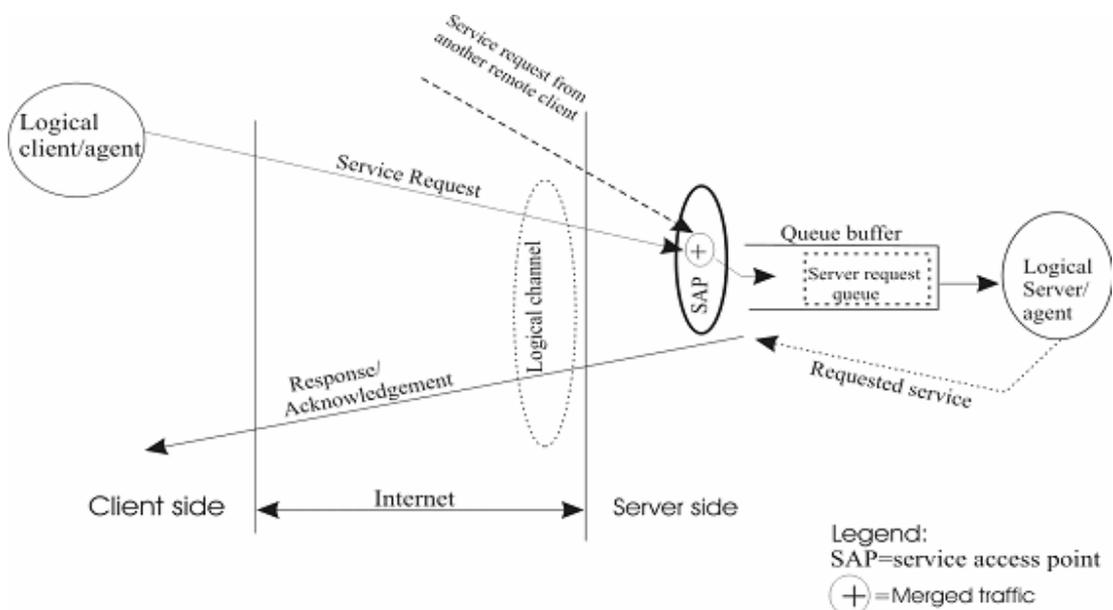


Figure 1.2. Client/server asymmetric rendezvous over an Internet TCP channel

To prevent buffer overflow various methods have been proposed for both the system/router level (e.g. [Braden98]) as well as the user/server level (e.g. [GAC02]). The success of a method depends on its capability of adapting to the changing pattern of the merged traffic [Paxson99] in two senses: a) the incoming request rate and b) the inter-arrival times (IAT) among the requests. In fact, traffic merging at the

server's SAP (Figure 1.2) is a stochastic process in which the IAT values form a time series [Molnar99, Paxson95, Taqqu03]. The IAT traffic pattern is unpredictable because it is decided by the network dynamics. For example, it may change suddenly from LRD (*long-range dependence*) to SRD (*short-range dependence*) or multi-fractal. Such changes can cause buffer [Holt00, Crovella97] and even system failure [Paxson95]. The SDITPM has the capability to leverage the traffic pattern as a primary metric, in a manner similar to others (e.g. server's queue length). It achieves this by traffic-independent, on-line and statistical direct data measurement that works with the *Convergence Algorithm* (CA). The CA is an IEPM (*Internet End-to-End Performance Measurement* [Cottrel99]) technique [Wong01], which is distribution/waveform independent, because it is derived from the Central Limit Theorem [Jain91]. In this case the CA is realized as the M^3RT Java API [Ip03].

The negative impact by IAT traffic patterns on the performance of object-based computing can be visualized by looking at the average number of trials to get a successful transmission in a client/server interaction. If the ξ is the probability error that causes retransmissions and the probability of achieving success at the j^{th} trial is $P_j = \xi^{j-1}(1 - \xi)$, the average number of trials (ANT) to achieve success is

$$ANT = \sum_{j=1}^{K \rightarrow \infty} jP_j = \sum_{j=1}^{K \rightarrow \infty} j[\xi^{j-1}(1 - \xi)] \approx \frac{1}{1 - \xi}. \text{ Therefore reducing } \xi \text{ would yield a}$$

smaller ANT and thus overall program execution time, as exemplified by the M/S paradigm in Figure 1.1. The SDITPM contributes to relieve congestion by load balancing and thus reduces the chance of server queue buffer overflow due to increased ANT. The problem is that a server in a busy node has to wait a long time

to virtually own the CPU and run again. During the waiting period its queue of incoming requests may be inundated to seriously overflow.

Figure 1.3 shows a real-life heavy-tailed RTT distribution, which was recorded for the TCP channel between the Hong Kong PolyU Intranet and the Hong Kong Star website over 24 hours. In the morning the typical channel RTT is 6ms but 154ms in the afternoon. The causes of this shift include: a) a longer queue for the Hong Kong Star web server and b) increased traffic in the network in the afternoon. If the website does not redirect the service requests to other collaborating servers or relocate the server to a less busy node, the ensuing widespread timeouts and retransmissions could congest the network easily.

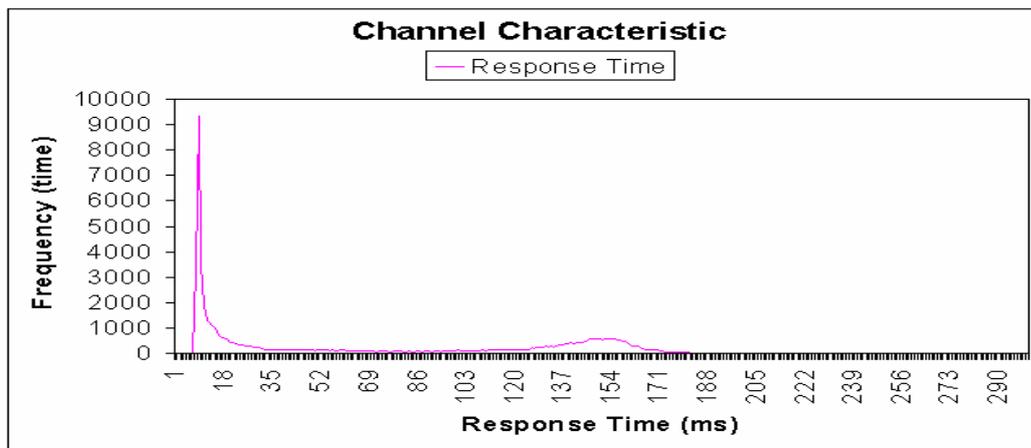


Figure 1.3. A real-life heavy-tailed RTT distribution over 24 hours

This kind of congestion easily propagates to other parts of the network and creates hot spots. A solution for this problem is to employ the “*split/migrate and re-direct (SMR)*” strategy, which includes the following options:

a) *Replication and request redirection*: The server agent replicates itself either fully or partially and then lets the replica migrate to another host. What follows is redirecting part/all of the request traffic to the replica.

b) *Self-migration*: The server agent migrates itself to a less loaded and/or faster node. The successful execution of these options, however, needs the support of agent mobility and proper policies as follows:

a) *Transfer policy*: The net effect is to decide if a server/agent migration is necessary and if the decision is affirmative then the *location policy* will be invoked.

b) *Location policy*: The net effect is to search for a suitable target host to house the migrating server, and the migration is realized only after a suitable target node is found.

The aim of the novel SDITPM framework proposed in this thesis is mainly on how to assist objects make sound migrating decisions. Therefore, the framework is basically a transfer policy decision making process for object migrations over the Internet. The necessity for a server to migrate hinges upon the SDITPM's steering power. A migration decision is affirmed for the $TP_o > Th_o$ condition, where TP_o is the overall transfer probability and Th_o the given threshold.

Object mobility usually involves different policies [Shirazi95] such as *information, transfer, selection, location* and *negotiation*. Information policy focuses on collecting system state information; transfer policy determines if an object should migrate; selection policy determines which logical object should migrate; location policy decides the destination node for a migration; negotiation policy handles the actual process of migration. In SDITPM the information policy is absorbed in the

metrics leveraging mechanism of the transfer policy decision making process. Since the location policy is not a focus in the thesis work but rather a way to assist the verification of the SDITPM framework, it absorbs all the details for the supposedly separate selection, location and negotiation policies into a single entity. In the SDITPM verification exercise the server mobility is monitored and traced by using the A^4 visualization tool proposed as part of this thesis (to be explained later). Once the SDITPM decision is affirmative migration would take place with the node of the lowest TP_o in the *Public Intranet* (PI) as the target. The meaning of PI will be elaborated later.

1.2 Problem Statement and Objectives

It is not easy to harness the service roundtrip time (RTT) in a client server interaction especially for a popular server because of the asymmetric rendezvous (one-server-to-many-slaves relationship). One way to shorten the average RTT of a logical server is load balancing through object migration. In the context of this thesis the object to migrate is the logical server, which automatically relocates itself from an overloaded host to a less congested one. The meaning of an overloaded server is usually well defined because it depends not only on the host's loading conditions but also the server's running priority. Therefore, it is important to leverage appropriate metrics to confirm "*if the host is congested with respect to the logical server concerned*". Profiles such as memory consumption and CPU utilization can reflect the general loading condition of a host but not the position of the server. One useful metric that reflects if a logical server is disadvantaged is its own average context

switching time (CST). Like everybody else the logical server should wait for its turn to virtually own the computer temporarily. This gives rise to two views for the context switching time. The first is the system view, which is concerned with the average CST for all tasks running in the system (i.e. the “intrinsic” CST). The second is the user view, which means that the logical server is interested only in its own average CST (i.e. the “self” CST). If the self CST is consistently much longer than the intrinsic CST, the likelihood that the logical server is disadvantaged is strong. As a result the server’s service RTT would be long. A viable solution to resolve this problem is to let the logical server migrate somewhere else.

Yet, if the server’s self CST is comparable to the host’s intrinsic CST but the memory and CPU utilizations are extremely high, then the server should also migrate to avoid operating under overloaded conditions. Logical server migrations, however, should be properly guided and steered in order to reap benefits from the load balancing effect that arises naturally from server mobility. The aim of this thesis is to propose a novel real-time steering mechanism. This mechanism should be incorporable as part of a logical server construct. It should leverage the chosen metric(s) on the fly. The leveraged metric(s) is quickly converted into an overall transfer probability TP_o that reflects the loading condition of the host. The TP_o value provides the basis to decide if migration is an appropriate option. In reality migration can only be materialized if a suitable target node is found by the location policy. The steering mechanism to be proposed in this thesis is at the transfer policy level, and the objectives to be achieved are as follows:

- a) Identify some useful performance metrics to support sound migration decisions made by logical servers in a real-time manner.
- b) Propose a technique (s) to efficaciously leverage metrics on the fly.
- c) Propose a scalable, non-intrusive conceptual framework that works at the transfer policy level to help logical servers make sound migration decisions; non-intrusive means no physical modifications of hosts are required.
- d) Verify the proposed conceptual framework.

Explanations of the terms mentioned are as follows:

a) *Direct data measurement*: It means that computations and/or decisions are based on data sampled on-line for the current time window. This approach has practical importance for large size networks because the profiles of any metric changes over time. To put this into perspective by using the Internet traffic as an example, the stability of any system running on the Internet can be seriously affected by different traffic patterns (i.e. inter-arrival time patterns). These patterns may even cause system failure if the system is designed with a preconceived mathematical model (e.g. Poisson) in mind [Paxson95]. It is a recognized fact that the Internet traffic could change from SRD (*short range dependence*) to LRD (*long range dependence*) and vice versa without warning [Lin05a]. Direct data measurement helps real-time applications decide statistically the behaviour (e.g. a specific traffic pattern/distribution) of direct parameters [Ip03]. It makes the accuracy of the result independent of the type of distribution/waveform (e.g. Poisson, binomial, heavy-tailed or self-similar [Molnar99]).

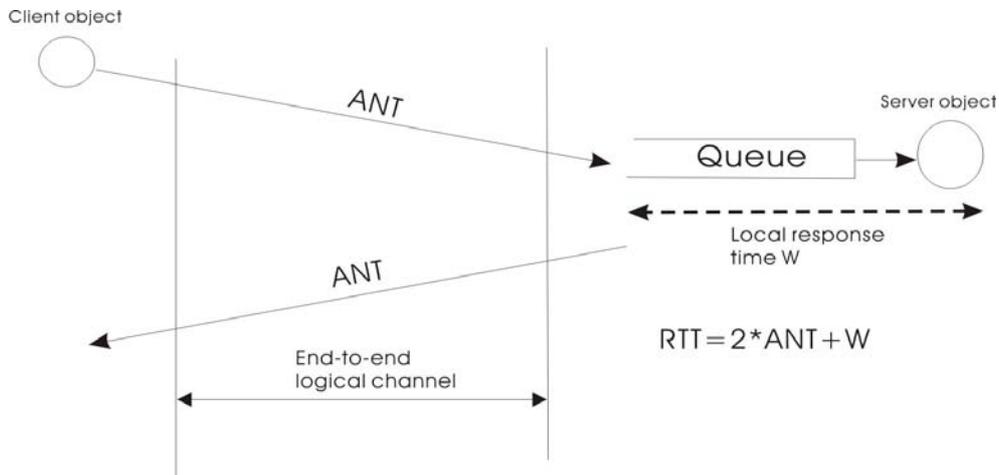


Figure 1.4. The RTT of a logical channel

b) *Object-based*: The software is made up of independent, cognate, collaborating logical entities known as objects or agents. These autonomous objects interact in an end-to-end manner over logical channels as shown in Figure 1.4. Applications over the Internet are naturally object-based and distributed [Lewis96]. The objects interact by message passing in a client/server relationship. The service time in this relationship is defined by the average number of trials (ANT) to get a successful transmission. The service RTT or response time is $RTT = 2 * ANT + w$. It includes the following time delays: communication, queuing, and service time. The communication delay can be absorbed into the average number of trials or ANT to get a successful transmission, which depends on the channel error probability ρ to

yield $ANT = \sum_{j=1}^{k \rightarrow \infty} j[\rho^{j-1}(1-\rho)] \approx \frac{1}{(1-\rho)}$. The queuing and service time delays at

the server side yield the response delay w . By assuming M/M/1 (M for Markov [Mitrani98]) operation, w can be defined by the equation,

$w = L * (1/\mu) = [\delta/(1-\delta)] * (1/\mu)$, where δ is the system utilization of the $(\lambda/\mu) \leq 1$ requirement for system stability. The parameters, λ and μ are the inter-arrival rate at the queue and the service rate by the server respectively.

c) *Real-time computing*: In real-time computation timeliness is the deciding factor for success. For an object-based real-time application the duration for the inter-object “*request/response*” cycle or roundtrip time (RTT) affects the overall execution deadline [Stankovic98]. Once the deadline has passed the computed result could be detrimental (for *hard* real-time applications) or meaningless (for *firm* real-time applications). Therefore, all the subtasks of a real-time application should interleave so that the execution time is within the deadline. Unfortunately, the logical channels over a large network (e.g. Internet) are usually plagued by dynamic faults that lead to widespread retransmissions, which are difficult to harness [Wong99]. If a channel has an average error probability ρ , then the average number of transmission trials (ANT) to achieve success is $ANT \approx 1/(1-\rho)$.

d) *Object mobility*: This is an important feature for successful contemporary distributed computing. It is the basis for the contemporary Remote Programming (RP) paradigm, as exemplified by the different mobile agent platforms (e.g. Aglets by IBM [AGLETS] and Telescript by General Magic [Cockayne97]). The core of the paradigm is to let objects/agents migrate freely. This is especially important if the agents can migrate to the locations of large databases and process the data locally. Since the objects send back the results (e.g. in data mining [RDM00]) after they have finished processing the in-between inter-object communication costs can be

greatly economized. This reduces network traffic congestion and frees more backbone bandwidth for sharing [Wu99]. Naturally mobility supports contingent object survival (known as persistence) and evens out processors' workload (load balancing).

Proper object mobility, however, needs the support of different policies, namely, *information*, *transfer*, *selection*, *location* and *negotiation*. Information policy is concerned with collecting system state information; transfer policy determines if an object should migrate; selection policy decides which logical object in the host should migrate; location policy selects the target remote node for a migration; negotiation policy ensures smooth migration. The leveraging primary metrics to support sound SDITPM decisions, in effect, fall in the information policy domain. The selection and negotiation policies are absorbed into the location policy in the SDITPM verification exercise because the SDITPM operates at the transfer policy level and the focus of this research is therefore mainly on this level.

1.3 Choice of Research Methodology

Object-based real-time computing is a relatively new and exploratory area [Oda01, Xu00, Corradi00]. In fact, a whole annual conference, *The IEEE International Symposium on Object-Oriented Real-time Distributing Computing (ISORC)*, is dedicated to resolve the related problems. For example, it was found that the traditional CORBA architecture cannot deliver the required timeliness in real-time computing and therefore new CORBA features are being explored with respect to this issue. This research started with surveys in different related areas and the aim

is to find a synergistic foresight for a novel measurement-based approach to support object mobility efficaciously.

In the realm of this research, three basic types can be identified as follows:

- a) *Exploratory*: It involves tackling a problem that is little known. The research would examine what concepts and theories that are appropriate to resolve the problem. The result of the research usually pushes the frontiers of knowledge.
- b) *Testing-out*: The aim is to find the limits of those previously proposed generalizations.
- c) *Problem solving*: It starts with a particular real-world problem and brings all the available intellectual resources to form the solution. The problem has to be defined and the method of solution has to be discovered.

With respect to the above classifications, the nature of this research is composite because it is problem solving that needs exploration as well. The methodology to support the exploratory process has to be discovered because it is not appropriate to adopt either the pure *top down* or *bottom up* approach. There are many seemingly related issues to be investigated and combined in a heuristic manner because the information in the field is mostly theoretical and experimental. After careful analysis the IRM (*Iterative Research Methodology*) as shown in Figure 1.5 is proposed. The IRM serves as the roadmap of research and allows repetitive heuristic backtracking toward the final solution. Further changes for this roadmap in the

future for more fruitful research is natural because the experience accumulated in the research process would re-orient some work.

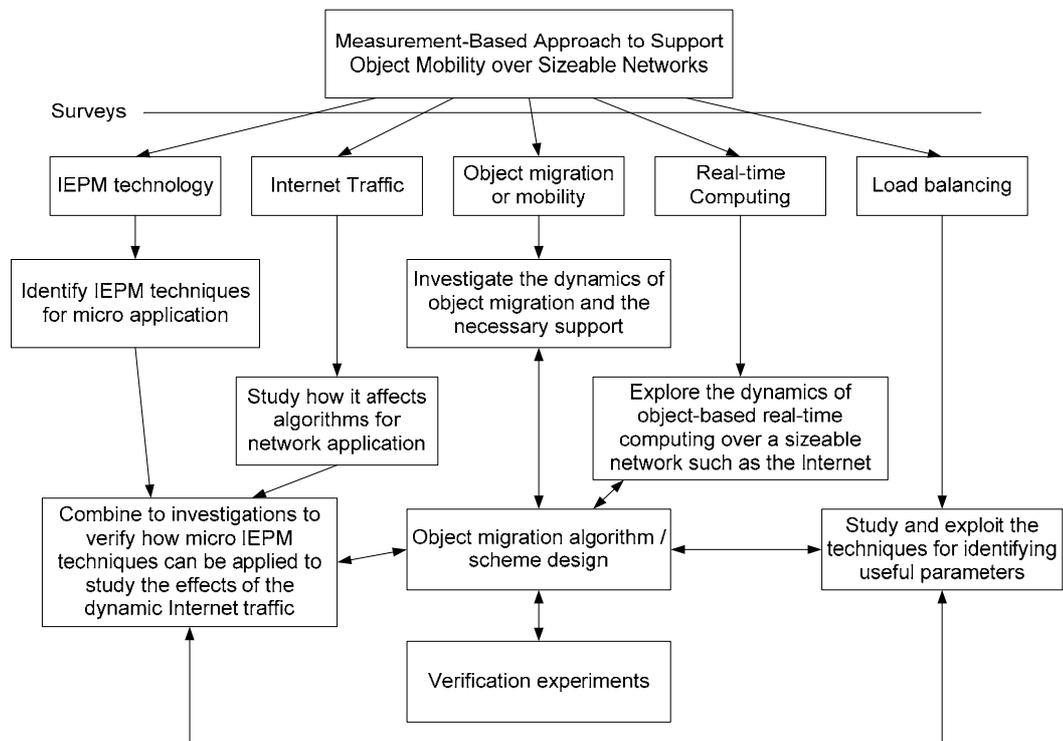


Figure 1.5. The Iterative Research Methodology (IRM)

1.4 Outline of the Thesis

This thesis is divided into six chapters as follows:

- a) Chapter 1 - Introduction: It concisely describes the importance of having an average short client/server service RTT for time-critical applications running on the Internet. For this reason direct methods have been proposed to shorten the service RTT directly or indirectly. Lengthy RTT is the result of lesser degree of overlapped parallelism. A solution to prevent system congestion,

which lengthens the service RTT, is the “*split/migrate and redirect (SMR)*” strategy. This strategy, however, needs the support of object mobility, which naturally produces the load-balancing effect. In this thesis the novel statistical distribution independent transfer policy model (SDITPM), which makes migration decisions at the transfer policy level, is proposed. It can be included as part of a logical server construct so the latter can be steered to make sound migration decisions that shortens the service RTT as a result the natural load balancing effect due to object mobility.

- b) Chapter 2 – Background and Related Work: A concise review of two modes of load balancing strategies: control and bidding. If the logical entities being managed are static, then it is the “*1-dimension static processing model*”. If the logical objects have mobility and physically migrate from one Internet node to another, it is called the “*2-dimensional dynamic processing model*”. Therefore the SDITPM framework belongs to the latter type. Since the leveraging of the chosen metrics by the SDITPM should be independent of the waveform/distribution, the Convergence Algorithm (CA), which is a real-time direct data measurement technique based on the Central Limit Theorem, was adopted. In fact, the meaning and achievement of mobility for logical entities has been evolving with time and programming paradigms, for example, from process migration to object migration.
- c) Chapter 3 – The Novel SDITPM Framework: The details of the framework are elaborated in light of the fact that it is a PID (P for proportional, I for integral and D for derivative) controller, which works with the deviation

errors and two parameters: primary and secondary. The meanings of planar and vertical integrations in the light of the SDITPM operation are defined.

- d) Chapter 4 – Verification Experiments: There are two types of verification experiments, static and dynamic. The aim of the static verification exercise, which involves only one Internet node and one logical server (an aglet (agile applet)) in the simulations in the Aglets mobile agent platform environment, is to make sure that the SDITPM framework is logically correct. In the simulations the logical server that has included the SDITPM mechanism as part of its construct was excited by different known waveforms (e.g. Poisson and self-similar) and pre-collected traces of different system profiles (e.g. CPU utilization). The chosen waveforms ensure that the SDITPM works correctly under known excitations. Only then the results produced with the pre-collected traces, which may embed different waveforms that are interleaved in the duration of the trace sampling process, are meaningful. The second stage of the verification exercise is dynamic and the logical object being monitored and steered by its SDITPM element actually migrates. Since object mobility monitoring needs a visualization tool that deciphers if the mobility correlates with the current statistics of the metrics being monitored and the corresponding migrations decisions, a reliable tool is needed. The detail of such a tool, which is part of the original work in this thesis, and the experimental results are presented in the next chapter focally.
- e) Chapter 5 – Dynamic SDITPM Verification and Visualization of Object Mobility: This chapter presents how the novel A^4 visualization package used

to support the dynamic verification of the SDITPM framework was derived from the previous A^3 framework. Different experimental results that leverage a single metric and multiple metrics to make migration decisions are presented for demonstration purposes.

- f) Chapter 6 – Conclusion: The verification results confirm that the novel SDITPM framework can indeed leverage different waveforms/metrics to support sound migration decisions at the transfer policy level. The experimental results indicate that SDITPM is one appropriate direction for achieving load balancing by object mobility over the Internet. Through guided object mobility shorter RTT can be attained and this makes the Internet more usable for serious time-critical applications, which require logically correct solutions to be ready before the deadline. The proposed immediate future work for the SDITPM research is to validate the framework with different real-time Internet based applications. By evaluating the validation results further possible improvements for the SDITPM framework may become apparent.

CHAPTER 2. BACKGROUND AND RELATED WORK

The sheer size of the Internet means an overwhelming number of possible network faults that affect the client/server RTT. If these faults are encapsulated by the error probability ρ , then the *average number of trials* (ANT) by a client to get a

transmission success is $ANT = \sum_{j=1}^{N \rightarrow \infty} jP_j$. The probability P_j for a success at the j^{th}

transmission is $\rho^{j-1}(1-\rho)$, and it yields $ANT \approx \frac{1}{1-\rho}$. Obviously reducing ρ

implies both lower ANT and RTT for better response timeliness. If the ρ_R is a component in ρ to represent the probability for retransmission, it should include the following errors: a) buffer overflow at the server side (user level [GAC02]), b) buffer overflow at the system/router level [Chatranon04], c) spurious timeouts due to an excessively long queuing time at the server side, (i.e. queuing time is longer than the client's timeout window), d) spurious timeouts caused by an unreasonably long routing time, and d) message loss due to partial hardware failure or random drop [Lakshman97]. To prevent buffer overflow from causing network congestion at the system/router level the IETF (*Internet Engineering Task Force*) proposes to adopt the AQM (*Active Queue Management*) approach in the RFC 2309 [Braden98]. Any AQM method works in two steps. The first is to throttle the sender to slow down transmission voluntarily. The second is to drop the incoming packets if throttling fails. Therefore, AQM methods and algorithms shorten the RTT of those already queued in the router buffer before the throttling on the expenses of those requests dropped later. They, however, do not eliminate those message retransmissions originated at the user/server level because of the merged traffic of the asymmetric

rendezvous (Figure 1.2). Actually there are many methods that reduce or eliminate the chance of user-level buffer overflow directly or indirectly. As a result the RTT is shortened because of less retransmissions by the sender. For example, if a logical entity is relocated to an idle node, then its waiting time to virtually own the CPU is reduced. Subsequently it processes its service requests faster and indirectly reduces the chance of user-level buffer overflow. The following are some methods that can shorten the client/server service RTT either directly or indirectly:

- a) *Dynamic buffer tuning*: This is achieved by adaptively adjusting the buffer length on the fly so that it always covers the queue length [GAC02, PID01] to eliminate user-level buffer overflow. The direct net effect is lower ANT and thus shorter client/server service RTT.
- b) *Adaptive timeout window*: It reduces the chance of spurious timeout by directly and timely changing the window size, for example, the “*adaptive timeout window*” strategy proposed Jacobson [Tanenbaum03]. The net effect is to directly reduce the ANT and shorten the client/server service RTT.
- c) *Traffic-sensitive transmission*: The aim is to sense the traffic pattern and react accordingly. The CMT (*Consecutive Message Transmission*) scheme [Wong99] is an example that deals with heavy-tailed traffic specifically. If the RTT is on the heavy tail, the CMT sends a few copies of the same request immediately one after another. The objective is to enhance the chance for the message to

be received earlier to shorten the RTT. The net effect is shorter client/server service RTT because it can avoid a lengthy routing process due to network hotspots. The potential problem is that the system can be congested with transmissions that quickly consume the network bandwidth and results in poor system throughput.

- d) *Logical entity migration* [Baratto00, Fuggetta98, Bivens99]: If the retransmissions are due to long queuing times because of an overloaded server, two solutions are possible. The first is to migrate the logical entity (e.g. server) to a faster and/or less busy host to benefit from the lower system utilization σ defined as $\sigma = \lambda / \mu$, where λ and μ are the average input (request) rate and the mean output/service rate of the server (capacity) respectively. The second is to clone the server and then let the clone(s) migrate to achieve distributed parallel processing. Conceptually this approach yields the speedup $S = (1 - \sigma/n) / (1 - \sigma)$, where n is the number of clones [Lewis96]. The third is for the object to migrate itself to a less busy node. The net effect of logical entity migration is shorter service RTT because it has more chance to virtually own the CPU and serves to empty his request queue quickly.

- e) *Load balancing*: This has been an important issue to address for efficient distributed computing. The aim is to have the workload evenly distributed among all the processors in the system. The net effect is that a logical entity can own the CPU more frequently to

reduce its request queue length. There are two basic approaches to achieve this aim: central control (i.e. sender-initiated) and bidding (i.e. receiver-initiated). In the central control mode the “coordinator” actively checks the workload of all the processors from time to time and allocates more loading to those seemingly less busy ones. In the bidding mode, however, the coordinator simply maintains a list of jobs/tasks to be executed. Those processors that are idle would bid for more work. It was found that the bidding mode usually yields higher performance/throughput, and some of the distributed systems are designed with the bidding principle. For example, the once popular Linda [Gelernter92], which is a coordination language and a distributed system working with a “*tuple space* (TS)”, works efficiently with bidding. It faded later because of the TS scalability problem over the Internet. Some researchers tried to resolve this problem with the concept of “distributed shared memory” but in vain. The Linda model, however, is a good example, of how bidding could potentially be used to improve the throughput of a distributed system. In the context of this MPhil research the Linda model is called “*static processing model*”, which is “1-dimensional”, because the processors (physical or logical) do not move. Instead they either wait passively to execute a new job coming from the coordinator of the control mode or solicit and pick up a new job actively in the

bidding mode. If the logical processors can migrate to idle nodes and follow the control or bidding mode of operation as well, this is the “2-dimensional dynamic processing model”. The second dimension is characterized by the process/object mobility that resolves many problems other than load balancing as illustrated in

Table 2.1.

Issues/problems addressed by object mobility	Concise description
System security (supported by both process and object migrations)	The logical entity migrates to a node of higher security upon detection of different problems such as the following: high frequency of viral attacks, and to many successful attacks by hackers. This is especially important for specific system such as those for banking purposes.
System safety (supported by both process and object migrations)	The logical entity moves out of dangerous zones such as those of high seismic activities, frequent volcanic eruptions, and potential fire hazards for self preservation or persistence.
Remote Programming Paradigm (RPP) (supported by object (also called agent) migration)	The aim is to let the logical entity go to the Internet site where the data is. The entity processes the remote data locally and sends back only the results. In this way the inter-object communication costs such those for RPC (remote procedure call) operations can be alleviated. The fringe benefit is more free Internet backbone bandwidth for sharing because of the reduced number of massive data transfers across the network. Normally e-commerce setups fall under the RPP realm [Cockayne97].

Table 2.1 Some issues/problems that can be resolved by process/object mobility

The success of the methods described above depends on how well the system dynamics is gauged. This means real-time sampling of chosen metrics such as CPU utilization, Internet traffic pattern, and buffer queue length are necessary. With real-

time knowledge of these metrics the system can proactively prepare to rectify coming problems in order to maintain a short service RTT on the fly. Dynamic buffer tuning is one such example, and it samples the queue length on the fly to ensure that the buffer length always covers the queue length adaptively [Lin05b]. The experience from [Lin05b] and [Paxson95] shows that the success of any real-time system should work with direct data measurement, which is based on real-time statistics. Therefore, the waveform independent method for real-time computation, namely, the Convergence Algorithm (CA) is an important technique for the SDITPM research.

2.1 The Convergence Algorithm

It is extremely difficult if not impossible to control the RTT of a TCP channel because of the unpredictability of the Internet traffic pattern. Therefore it is impractical to design a SMR solution based on a preconceived distribution (e.g. Poisson) [Paxson95]. Rather the solution should be based on direct data measurement, which uses the current statistics to determine the appropriate solution. For example, the *Convergence Algorithm* (CA), which is an IEPM technique [Wong01] at the micro level, is a powerful tool to assist real-time direct data measurement. Being micro the implementation of the technique exists as an object that can be invoked for service by message passing anytime and anywhere. The CA is derived from the *Central Limit Theorem*, and this makes its accuracy independent of any type of waveform/distribution [Ip03].

The *micro* CA (MCA) is a structural component of the novel SDITPM framework proposed in this thesis, but it operates as an independent object parallel

with the SDITPM main body. The Java-based implementation of the MCA is known as the M^3RT [M^2RT , Ip03]. The CA essence is captured by the equations: (2.1) and (2.2). M_i is the predicted mean from the data samples collected in the time window of the i^{th} prediction cycle. The actual window size is determined by the total time needed for collecting the $(F - 1)$ data samples for the prediction. The F value is the *flush limit*, and the past experience indicates that $F = 14$ yields the fastest convergence [Wong01]. The other parameters include: a) M_{i-1} as the feedback of the last predicted mean to the current M_i prediction to make the latter naturally cumulative/integrative, b) m_j^i is the j^{th} data sample in the i^{th} prediction cycle, for $j = 1, 2, 3, (F - 1)$, and c) M_0 as the first sample after the MCA had started running. Figure 2.1 shows the M_i prediction for a TCP channel that connected the LaTrobe University website in Australia and the Hong Kong PolyU. The *mean message response time* (M^2RT) or M_i changes in every prediction cycle responsively and reflects the current perturbations/changes embedded in the current data samples. The M_i in this example tends to settle to the steady state value of 480ms. The M^3RT (*micro* M^2RT) implementation (i.e. MCA) is suitable for real-time computing because it requires very short execution time. It differs from the CA's *macro* form, which is a tool that requires human intervention in the data collection and prediction processes.

$$M_i = \frac{M_{i-1} + \sum_{j=1}^{j=F-1} m_j^i}{F} \dots\dots\dots (2.1); M_0 = m_{j=0}^{i=1} \dots\dots\dots (2.2); i \geq 1$$

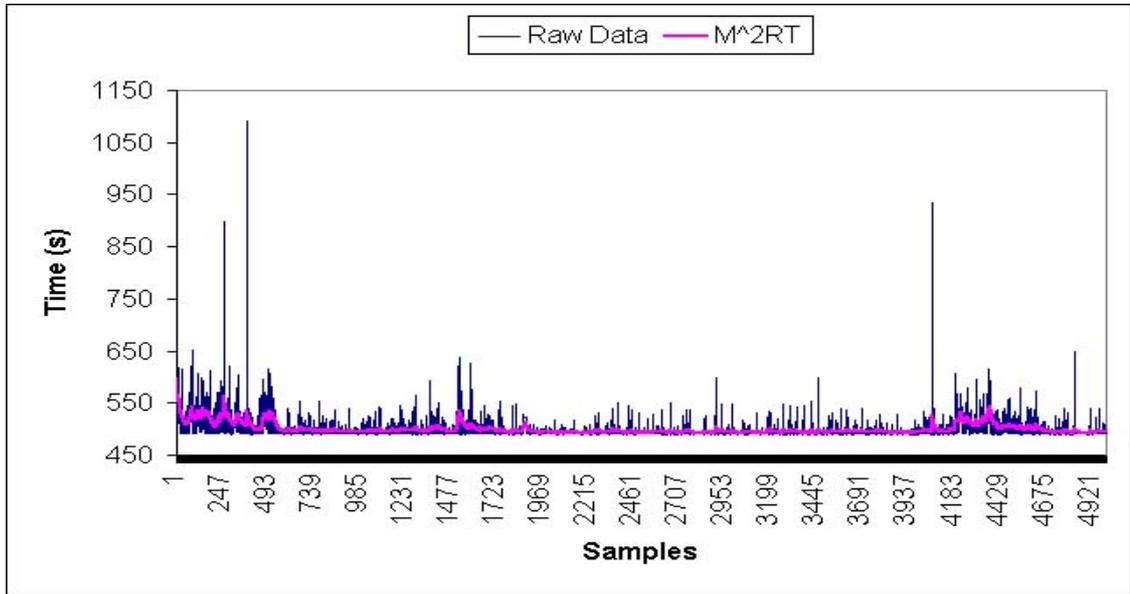


Figure 2.1. M_i measured by MCA for the TCP channel between Hong Kong PolyU and LaTrobe University in Australia

2.2 From Process Migration to Object Migration

The scope of this research is measurement-based load balancing by object mobility. In fact, object mobility has come a long way from process mobility. The mobility from [Goscinski91] process migration is usually initiated by the host of the process. The host initiated the necessary procedures/policies such as information, selection, transfer and location. As the paradigm for programming was evolving to match new technologies such the advent of Internet process migration no longer suffice. One particular problem with process migration and the interim solutions is code compatibility. In the beginning the compiled object coded followed the migrating process. This caused problems and failure because the compilers and the

hardware in different network nodes may not be compatible. The object-based programming paradigm helped resolve this problem because the object classes rather than the compiled object code modules followed the migrating process. The classes are instantiated in the target machines and they should be compatible with the local hardware. The power of such instantiations was demonstrated clearly by the object-oriented RHS (Reciprocal Hypercomputer System) distributed computing framework [Wong96]. This framework is characterized by the following salient features: a) the classes can be written in different programming or natural languages, b) the classes are treated like hardware “machine parts”, c) the classes follow the migrating logical object (not a process) and are instantiated at the target machine, and d) the presence of a coordination system similar to Linda enables different interacting objects in the system to interact. In fact, the RHS is among the first of its kind to separate object and process as two ingredient entities in distributed computing [Chin91]. As it was pointed out by [Cockayne97] the drive for successful e-commerce on the Internet needed the Remote Programming Paradigm (RPP). For example, the user can dispatch an intelligent object to achieve a goal over the web (e.g. search for the cheapest named book). This object is now called agent because its function is similar to a human agent who is delegated a duty, for instance finding and buying the cheapest X book from among the many web based bookstores (e.g. Amazon). Those platforms design to achieve the RPP purposes are called mobile agent platform (e.g. the IBM’s Aglets and the Mitsubishi’s Concordia system).

The appearance of different mobile agent platforms enable researchers to investigate how object mobility can benefit distributed computing in general.

Among the first batch of researchers to investigate the totality of object mobility, distributed software debugging, and program visualization, Wong et al [Wong00] successfully proposed the A^3 framework for deciphering distributed software problems by visualizing abnormal object mobility/behaviour. Besides, object behaviour can be logged for detailed analysis later. The experimental results presented in the paper were obtained by inducing object mobility through load balancing. The master/origin of the mobile object/agent, the migrating object itself and the current host all computed different migrating indices for the agent. If any one of the three indices exceeds the given threshold, the agent is warned of the imminent danger and advised of migration to another node as an option. The final decision of migrating or not rests with the agent itself. The way that the migration indices were calculated in [Wong00] is equivalent to proportional (P) or scale control only. In light of load balancing the A^3 tool helps verify an algorithm through monitoring and visualization of agent behaviour [Wong00]. The fringe benefit of tracing object mobility is the identifications of congestion, hotspots and bottlenecks in the underlying network. In fact, the A^3 experience in terms of sudden network congestion for various reasons indicates the importance of load balancing by object mobility. This experience has inspired the SDITPM proposal and investigation.

The SDITPM, which is for application at the transfer policy level, is novel because it is based on real-time PID (P for proportional, I for integral and D for derivative) control that works by direct data measurement. The contribution by the SDITPM framework is significant because it help shortens the client/server service RTT and makes the Internet more usable for object-based real-time applications.

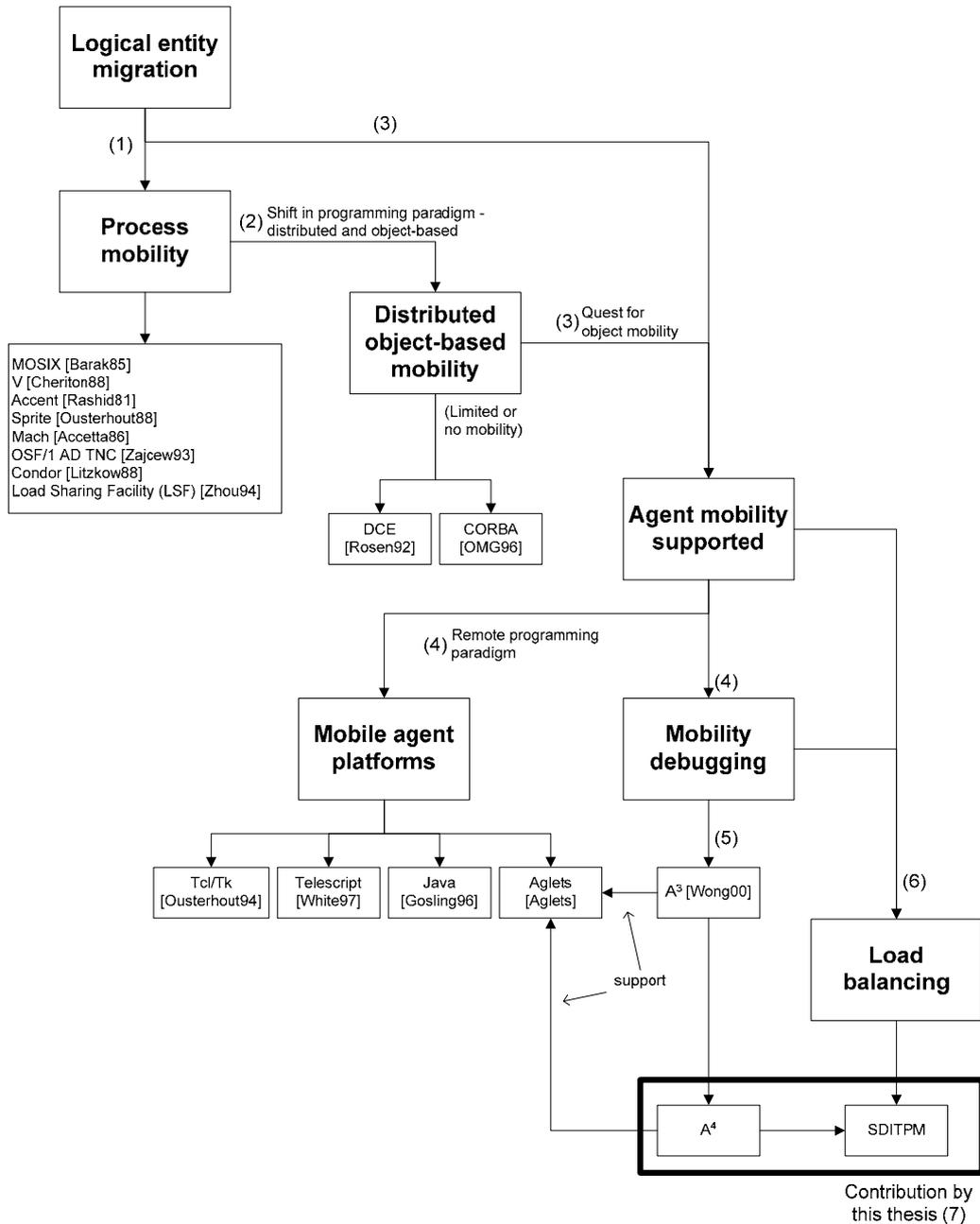


Figure 2.2. Evolution of logical entity migration

Figure 2.2 summarizes the evolution from process migration to object migration and the novelty of the SDITPM framework. Logical entity migration encompasses both process and object migrations. The marker “(1)” indicates that

logical entity migration started with process migration. As the programming paradigm had shifted from purely algorithmic to object-based so did distributed processing (marked by “(2)”). The DCE (Distributed Computing Environment) and CORBA are two architectures that support effective inter-object interactions independent of the physical locations. Later object mobility is a need for scalability, security and safety purposes and this started the quest to effectively support object mobility (marked by “(3)”). This quest brought about the Remote Programming Paradigm and the proposals of different mobile object platforms (e.g. Aglets) as marked by “(4)”. Meanwhile the problem of debugging object behaviour correctly called for the technique of program visualization (marked “(5)”). For example, the visualization tool A^3 [Wong00] worked well for the Aglets system. It was found that allocation of objects manually might not produce the expected effect because the loading conditions of different nodes changed quickly. This problem could be resolved by load balancing and object mobility, if properly leveraged, can produce the necessary load balancing effect (marked “(6)”). This thesis treated this pristine area of load balancing by proposing the SDITPM transfer policy framework, which can be integrated as part of a logical server. It helps the latter make sound migration decisions at the transfer level by leveraging some chosen system parameters. At the same time A^3 is extended to the novel A^4 , which together with the SDITPM framework, represent the contribution of this research.

2.3 Load Balancing in a Decentralized Computing Environment

Any effect that can even out the workload among different collaborating nodes in a distributed or decentralized computing environment is load balancing.

This effect is important to the success of Internet Service Providers (ISPs) because it shortens the service roundtrip time (RTT) for the customers. This satisfies the QoS requirement contracted to the users and makes them happy; “a happy customer is a return customer!” The prelude to effective load balancing, however, is successful network monitoring, which is an important network management issue still in the immature stage [Baratto00]. The fundamental of network monitoring is to sample chosen metrics in a real-time and non-intrusive manner. The sampling process should not at anytime introduce unnecessary perturbations to the network being monitored, and therefore the SNMP (Simple Network Management Protocol) is a popular approach because it is a de facto standard for network management.

In the SNMP architecture every node being managed runs a simple SNMP process called the SNMP agent for efficiency. This agent maintains a local database that registers the current states and history of its operation. Sophisticated tasks of network management are done by designated nodes called the *management stations* in the SNMP context. Each station has one or more entities that communicate with the SNMP agents to sample their statistics. The statistics should be as real-time by nature as possible in order to produce significant contributions in network monitoring and management. The SNMP stations usually have graphical user interface to facilitate the network manager to rectify situations. Although the original intention of SNMP is to facilitate network managers to intervene, the latter developments support migration decisions by mobile logical objects without human interventions. The core of this is the logical repository that contains all the updated state information and operation history of a network cluster [Bivens99, Fuggetta98].

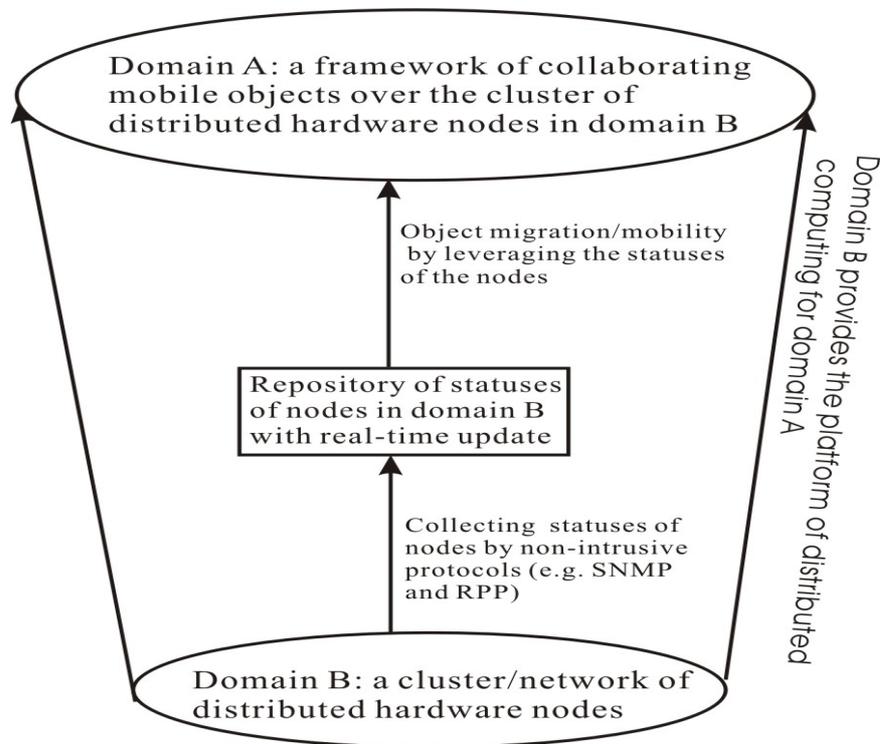


Figure 2.3. Concept of a repository for nodes' statuses

Successful load balancing in a decentralized, object-based computing environment depends on object mobility. An object, which has the minimum intelligence, leverages chosen metrics to make its migration decisions. In the context of this thesis, these decisions are made at the transfer policy level. Whether the decisions are realized or not depends on the location policy that determines if target nodes for the migrations exist. This is achieved by leveraging the metrics that are similar to those used by agents. Therefore, a repository that contains all the updated state information and operational history for all the distributed hardware in the network cluster should exist. Both the transfer policy and the location policy make their separate decisions by leveraging the same repository. Figure 2.3 shows the

repository concept, in which the nodes' operation statuses can be sampled by the SNMP and RPP (Remote Programming Paradigm) approaches. The RPP approach is more flexible because a mobile agent is used to carry out the job of the SNMP agent. The advantages is that the mobile agent can move close to any interested network location quickly anytime and samples the wanted information without necessitating massive information transfer across the network. That is, the object mobility reduces the chance of introducing unnecessary traffic to the system due to the network monitoring and management activities [Cockayne97].

In the distributed computing environment in Figure 2.3 a framework of collaborating logical objects (i.e. domain A), which interact by the client/server relationship, run on the distributed hardware in domain B. The logical objects leverage metrics recorded in the repository to make the necessary migration decisions. The information in the repository is updated continuously in a real-time manner by using different protocols such as SNMP and RPP that samples the nodes' statuses in domain B. Network monitoring and management for distributed environments as shown by Figure 2.3 are immature. The focus nowadays is how to achieve effective RPP based network monitoring to avoid those perturbations inadvertently introduced by the SNMP approach. These perturbations are caused by the massive information transfer across the network and communication activities between SNMP agents and the routers being polled. If the SNMP agents can be located very close to the routers or objects being polled at will, then the communication traffic can be reduced.

The SDITPM framework differs from Figure 2.3 because a logical server that embeds the SDITPM mechanism makes its migration decisions by directly leveraging the metrics of the current host. It has not considered the need of a repository.

2.4 Recap of Effective Distributed Processing

It is useful to recap the evolution of distributed processing so that the importance and novelty of the SDITPM framework proposed in the paper becomes apparent. There are four important elements in the evolution process, and these elements evolve in different speeds depending on the technological and conceptual advancement at the time. As a result the various degrees of delays make it difficult to achieve ideal distributed processing at the same time. The most fundamental issue of distributed computing is speedup to be gained from distributed parallelism. By Moore's Law the VLSI speed had already hit the limit of physics in the early 2000s [Lewis96]. The limit is due to the molecular distance of the silicon wafer. It is fortunate that the appearance of pure fiber optics that work with solutions can overcome the problem of molecular distance and double the network bandwidth roughly every 14 months. Eventually in an asymmetric rendezvous (i.e. the many-clients-one-server relationship) the logical server is easily inundated by the avalanche of incoming requests. The logical solutions to overcome this problem are as follows: a) the server redirects requests to other independent servers in the network and the resultant speedup due to load balancing shortens the average RTT, b) the server clones itself and relocates the clones physically to achieve the same load

balancing effect as above, or c) the server migrates to a much faster and idle node to achieve the same speedup and RTT shortening effects. In fact, the intrinsic SIMD (single instruction and multiple data) parallel architecture, which works by lock-steps, is the ultimate goal of distributed computing, which was exemplified in Figure 1.1. The concept is that overall program execution time T_{PE} involving N distributed objects that collaborate asynchronously is equal to $T_{PE} = RTT_{average} * P * N^{e^{-\beta}}$, where P is the number of passes to complete program execution and β is the degree of overlapped parallelism. For a very large β , which indicates 100% parallelism, the net effect is SIMD operation but on a network; that is, $T_{PE} = RTT_{average} * P * N^0 \approx RTT_{average} * P$. As a matter of fact $RTT_{average}$ correlates with β because a high $RTT_{average}$ usually implies the following: a) long context switching time for the logical server, b) long queuing time for a request to get server by the server, and c) long communication delay due to network congestions and hot spots along the routing paths. Effective load balancing can resolve these problems in one shot because a logical server can migrate to a fast idle node. Effective load balancing by logical entity mobility is still rudimentary because there is still little published experience. This can be deduced from the following chronicle:

- a) Process migration at the distributed operating system (OS) era [Goscinski91]: This concentrated on the Unix environment and in a distributed environment the collaborating nodes would have no problem of accepting a migrating process. The distributed OS basically neutralizes the incompatibility of the hardware

heterogeneity in the underlying network. The obstacle though is code incompatibility. In this era the process migrates with the compiled code, which may not run correctly at the target node. This problem is resolved by the object-based or object-oriented paradigm, where the migrating entity, which is always supported by a process (i.e. a virtual machine), carries along the object class. The executable code is instantiated/compiled at the target node and this ensures that the code runs correctly on the local platform.

- b) Interim approaches: They do not need the support of a uniform distributed operating system but rather a uniform middleware, which masks out the differences among the heterogeneous hardware and their peculiar OS. The principle is to form a distributed process environment (DPE) by the following steps: a) select a list of nodes to support the DPE and in the PVM (parallel virtual machine [PVM90, PVM93]) example this is called the “node list (NL)”, b) execute a PVM library command to lay the PVM middleware on top of NL nodes, c) manually relocate the different subtask instances, which are compiled codes, in different nodes, and d) organize the subtasks or logical entities to collaborate. The main benefit from the PVM or similar approaches (e.g. [P4]) is use of a middleware, which requires no modifications in the supporting hardware and OS platforms. The drawback is that they do not support load balancing by object mobility as a feature.

- c) Object-based Distributed Service Architecture (DSA): The success of the World Wide Web (WWW) and Internet make the concept of object-based computing even more appealing. Running distributed objects over distributed Internet hardware is natural and this is usually referred to as distributed object computing (DOC). In fact, DSA, DOC and DPE are synonymous. The main issue is how to harness the object behavior for efficient computation. To facilitate this many different contemporary frameworks and middleware were proposed (e.g. CORBA (Common Object Request Broker Architecture [OMG00]), IN [OMG97] and TINA (Telecommunication Information Networking Architecture [OMG97]) in different domains of applications. The performance problems of these frameworks are still open issues. For example, the TINA software architecture is aimed at supporting rapid service deployment in multivendor and multitasking environments. It was found by different simulations that “sender-initiated or control” load balancing can indeed improve TINA performance (e.g. [Kih197]). The assumption is that the routers would have the intelligence to redirect requests to different logical servers that can provide the same service. Achieving this, proper real-time network monitoring and management is theoretically required.
- d) Real-time network monitoring and management (RT-NMM): Network management is traditionally based on SNMP and requires

human intervention. In order to support load balancing in DPE environments the real-time approach is necessary. Up to this moment RT-NMM is still an open issue [Joo01], and the trend is to use intelligent agents to gather node statistics, put them in a repository and update them in a real-time manner [Thottan98]. The solution to the most fundamental issue remains elusive: “how do we ensure that the information stored in the repository is indeed current?” Using out-dated information for the sake of load balancing can be deleterious.

- e) Migration index for the DPE: In the paper [Wong00] the authors argued and demonstrated that object migrations should be the decision of the object itself even though the following should compute different migration indices by leveraging useful metrics: the object itself, the master or origin of the migrating object, and the current host of the object. The migrating object should be given the intelligence to check the cost and effectiveness of the migration act by comparing the three metrics. The migration decisions by an object are considered as the transfer policy level, and whether the decisions can be materialized or not depends on whether suitable destinations are found by the location policy. The same paper also points out that any DPE should be formed by public declarations and the example provided is the Public Intranet. Any logical entities that migrated into a managed DPE without a proper

membership privilege are regarded as intruders, and this provides the minimum security. The object migration scheme proposed in [Wong00] is generic and based on proportional (P) control only. Its effectiveness, however, has inspired the research of this thesis, which focuses on proposing a PID controller, SDITPM that supports real-time migration decisions at the transfer policy level. The SDITPM mechanism does not depend on the presence of a repository as shown by Figure 2.3. Instead it leverages chosen metrics as waveforms for making migration decisions on the fly. In this way, the SDITPM framework is platform independent and immensely suitable for Internet applications. The only requirement for the SDITPM mechanism to deliver effective load balancing that contributes to shorten the service RTT is a valid DPE such as the Public Intranet (PI), which will be explained in detail later.

2.5 Connective Summary

This chapter explains the different methods whereby the client/server service RTT can be shortened either directly or indirectly. Load balancing, which can be implemented as either a 1-dimensional static processing model or a 2-dimensional dynamic processing model, can achieve the purpose. The 2-dimensional model, which is the combination of “1-dimensional model + entity mobility”, is usually more efficient. The Convergence Algorithm (CA) that can leverage real-time statistics independent of the distribution type is elaborated. The literature search

indicates that there is no PID controller that can steer sound migration decisions by logical entities at the transfer policy level. In the next chapter a novel PID controller called the SDITPM framework will be proposed.

CHAPTER 3. THE NOVEL SDITPM FRAMEWORK

3.1 The Conceptual Framework

The SDITPM framework is generic and based on the concept of a system steady state [Mitrani87], which is generalized by the expression/function (5.1). The formal parameters in function f are primary metrics pm_x for $x = 1, 2, \dots, n$. If the state of every primary metric is represented by its objective function $\{0_{pm_x}, \Delta_{pm_x}\}^2$ [GAC02, Lo05a, Lo05b], then 0_{pm_x} is the steady-state reference and Δ_{pm_x} is the safety/tolerance margin about 0_{pm_x} (i.e. on either side of 0_{pm_x}). The “ $\{\}^2$ ” representation indicates the absolute nature of the value with the brackets $\{ \}$ being used. Ideally, if the system f is at its steady state, then all its 0_{pm_x} values should be the given references for the *primary metrics*. Practically, the system is considered at its steady state of operation provided that there is no deviation outside the fault tolerance band of $\pm \Delta_{pm_x}$ for any primary metric in f . These deviations are called *deviation errors* in the SDITPM context.

$$f(pm_1, pm_2, \dots, pm_n) \dots (5.1)$$

It is normal that the system dynamics, under the influence of different driving forces, would push some of the primary metrics in f outside the $\pm \Delta_{pm_x}$ bands. Then, the system should self-compensate [Ip01] and rein the reneged

primary metrics back to the corresponding safety bands. If the primary metric value at time t , namely, $V_{pm_x}^t$ is greater than $(0_{pm_x} + \Delta_{pm_x})$ or smaller than $(0_{pm_x} - \Delta_{pm_x})$, the difference is the *deviation error* (DE), which is equal to either $DE_+^{t,pm} = V_{pm_x}^t - 0_{pm_x} - \Delta_{pm_x}$ for $V_{pm_x}^t > (0_{pm_x} + \Delta_{pm_x})$ indicated by the + DE subscript or $DE_-^{t,pm} = (0_{pm_x} - \Delta_{pm_x} - V_{pm_x}^t)$ for $(0_{pm_x} - \Delta_{pm_x}) < V_{pm_x}^t$ indicated by the – DE subscript. The + or – subscript is decided with respect to whether DE is on the right or left side of 0_{pm_x} . If deviation error(s) appears, the system needs to take compensation measures to eliminate it because DE means that the system is no longer in its steady state of operation.

$$f[pm_1(sm_1, sm_2, \dots, sm_k), pm_2(sm_1, sm_2, \dots, sm_k), \dots, pm_n(sm_1, sm_2, \dots, sm_k)] \dots (5.2)$$

If the primary metrics in f are defined by their *secondary metrics*, for example as $(sm_1, sm_2, \dots, sm_k)$ as shown in equation (5.2), then the primary metric pm_x is, in effect, a function itself. The concept of deviation error holds for the secondary metrics. That is, the DE for a secondary metrics is either

$$DE_+^{t,sm} = V_{sm_x}^t - 0_{sm_x} - \Delta_{sm_x} \quad \text{for} \quad V_{sm_x}^t > (0_{sm_x} + \Delta_{sm_x}) \quad \text{or}$$

$$DE_-^{t,sm} = (0_{sm_x} - \Delta_{sm_x} - V_{sm_x}^t) \quad \text{for} \quad (0_{sm_x} - \Delta_{sm_x}) < V_{sm_x}^t .$$

From the pm (primary metric) and sm (secondary metric) perspectives, leveraging deviation errors in the SDITPM framework can happen in three levels as follows:

a) *Point level:* It is based on the delimiters, $(0_{pm_x} + \Delta_{pm_x})$ and $(0_{pm_x} - \Delta_{pm_x})$ defined for the primary metric. If each primary metric in equation (5.1) contributes a DE value as a point-level effect, the collective effect is a combination of them. If addition is the operator, then the collective point-level is conceptually defined by summation or integration, $\sum_{pm_x=1}^{pm_x=n} pe$ or $\int_{pm_x=1}^{pm_x=n} pe$, where pe is point-level effect per primary metric. In a generic sense finding the collective/combined effect from all the primary metrics being leveraged is vertical or incidental integration in the SDITPM context.

b) *Plane level:* It is based on the delimiters, $(0_{sm_x} + \Delta_{sm_x})$ and $(0_{sm_x} - \Delta_{sm_x})$ for the secondary metrics that collectively define the primary effect (of a primary metric). That is, the primary effect is the combination of all the secondary effects as deviation errors (from the secondary metrics). If the operator is addition, then the plane-level effect for every primary metric is $\sum_{sm_x=1}^{sm_x=k} se$ or $\int_{sm_x=1}^{sm_x=k} se$, where se indicates the effect per secondary metric and $\int_{sm_x=1}^{sm_x=k} se$ is planar integration.

- c) *Stack level*: This is the total combined/integral effect, when the secondary effects of many primary metrics are leveraged. Equation (5.1) represents the collective stack-level effect obtained by

vertical integration $\int_{pm_{x=1}}^{pm_{x=n}} [\int_{sm_{x=1}}^{sm_{x=k}} se]$ conceptually.

3.2 The Concept of PID Control by SDITPM

The SDITPM framework is conceptually a PID (proportional (P) plus integral (I) plus derivative (D)) controller, which makes sound migration decisions for logical entities at the transfer policy level. These real-time decisions made with metrics statistics sampled on the fly steer a logical entity to migrate efficiently and effectively from one Internet node to another. The result is improved serviceability of a logical entity (e.g. agent or server) in “wired + wireless” Internet environment. Serviceability is the “*chance of obtaining a required service within a defined period*”. Better serviceability is obtained because of the effect of load balancing arising from the guided object mobility. The SDITPM embedded in an object leverages chosen primary metrics to derive the secondary ones. For example, the P, D, and I control elements are secondary metrics, which can be timely combined by planar and vertical integrations. Vertical integration is a must to produce the stack-level effect when several primary metrics are leveraged at the same time. A migration decision depends on the overall transfer probability TP_o , which assumes the highest regional transfer probability TP_r . The migration decision is affirmed for the $TP_o > Th_o$ condition, where Th_o is equal to the given threshold Th_r of the

control region r (e.g. Region 1 in Figure 3.1) where the current TP_o value comes from. There are four control regions per primary metric being leverage by SDITPM: R1 for PID (i.e. “P+I+D”) control, R2 for DI (“D+I”) control, R3 for PI (“P+I”) control, and R4 as an inert of “*don’t care*” state. At any time only one control region will become dominant and its regional transfer probability TP_r becomes the TP_o to trigger a migration for the condition, $TP_o > Th_o$.

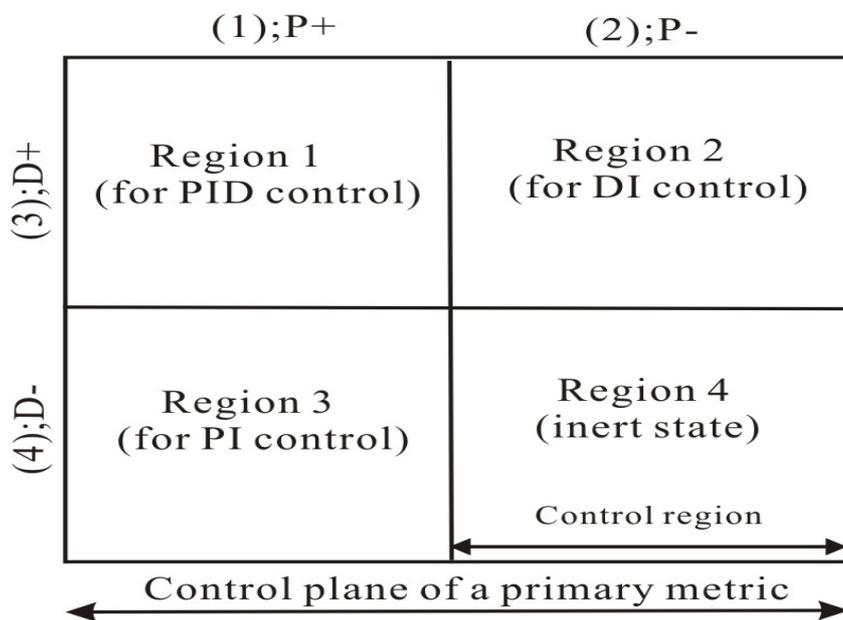


Figure 3.1. The four control regions of a primary metric

Figure 3.1 shows how the four control regions of a primary metric define its control plane. In the SDITPM framework, every primary metric contributes two secondary metrics, namely, P and D controls. The effect of each control region, which is defined by the $\pm P$ and $\pm D$ combination, is unique as follows:

- a) *Region 1*: It works with PID control and is alternatively called $r(1,3)$, r^{PID} , $r^{C_3^1}$, or C_3^1 in the SDITPM nomenclature.
- b) *Region 2*: It works with DI and is alternatively called $r(2,3)$, r^{DI} , $r^{C_3^2}$, or C_3^2 .
- c) *Region 3*: It with PI control and is alternatively called $r(1,4)$, r^{PI} , $r^{C_4^1}$, or C_4^1 .
- d) *Region (2,4)*: This inert state is also called $r(2,4)$, r^{Inert} , $r^{C_4^2}$, or C_4^2 .

Other than r^{Inert} the other regions in a control plane are continuously active. The current behaviour of a region in the control plane is characterized by its regional transfer probability TP_r , for example, $TP_{r(1,3)}$, which is the result of planar integration. The $TP_{r(1,3)}$ expression is conceptually equal to $\sum_{sm_x=1}^{sm_x=k} se$ or $\int_{sm_x=1}^{sm_x=k} se$ (i.e. planar integration), which sums deviation errors derived from the “ $P+$ ” and “ $D+$ ” values. For example, $P+$ at time t means $DE_+^{t,P} = V_P^t - 0_P - \Delta_P$ for $V_P^t > (0_P + \Delta_P)$ and $D+$ indicates $DE_+^{t,D} = V_D^t - 0_D - \Delta_D$ for $V_D^t > (0_D + \Delta_D)$.

The regional transfer probability threshold Th_r has two meanings (i.e. planar and incidental perspectives as shown in Figure 3.2). Firstly, it is the regional threshold when only one primary metric is leveraged by SDITPM (i.e. planar integration). Secondly, it is the stack-level threshold when more than one primary

metrics are leveraged (i.e. vertical integration, namely, $\int_{pm_x=1}^{pm_x=n} [\int_{sm_x=1}^{sm_x=k} se]$). In every transfer policy object migration decision cycle, the TP_r values of all the active regions are computed at the same time. Then, the large TP_r value becomes the overall system transfer probability; that is, $TP_o = \text{highest } TP_r$ or $TP_o = \max(TP_r)$. The migration decision is affirmative for $TP_o > Th_o$, where Th_o is either separately defined or equal to the Th_r of the active region where the current TP_o comes from.

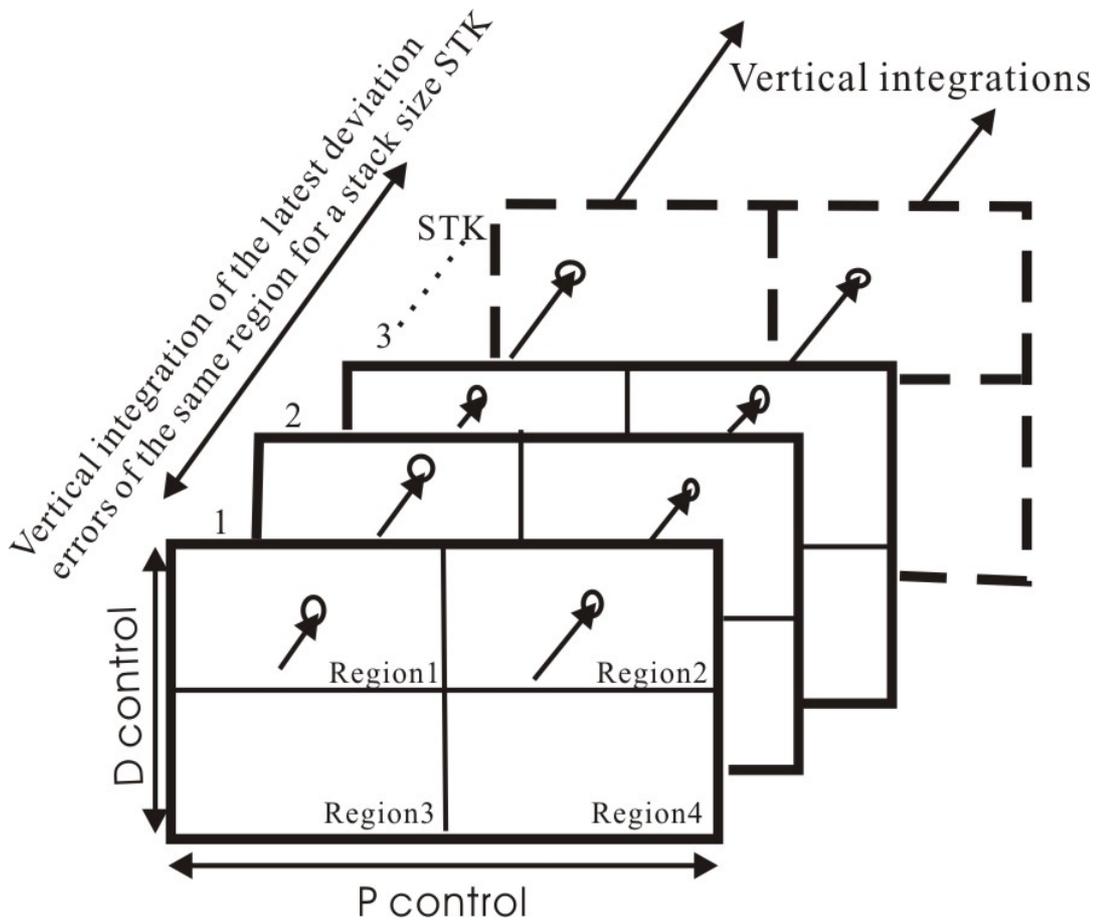


Figure 3.2. Leveraging multiple primary metrics by SDITPM

Figure 3.2 captures the PID essence of the SDITPM controller in leveraging multiple primary metrics. From each primary metric the P and D secondary metrics are derived. The planar integration produces the planar TP_r value for every region r in the control plane, and the vertical integration produces the collective TP_r value for a stack of r^{th} regions (e.g. $r = R1$). In Figure 3.2 the stack size (i.e. the number of r^{th} regions) is STK. In reality any useful system parameter can be leveraged as a primary metric. For example, if the request queue length (Q) of a logical server is leveraged as the primary metrics, then the secondary metrics derived are as follows:

a) the ratio M_i / M_{i-1} for P control, where M_i^t is estimated on the fly at time t (in the i^{th} cycle) by the Convergence Algorithm, and b) the rate of change in the same cycle, dQ/dt for D control.

If the ψ symbol represents deviation error, l to index the primary metric, d to index the secondary metric, $I_{\psi_{l,d}}^r$ or $I_{\psi_{l,d}}$ to denote the planar integration for region r , then the regional transfer probability $TP_{l=[1,n]}^r$ for leveraging n (i.e. STK = n) primary metrics can be scaled as required by $TP_{l=[1,n]}^r = \alpha[1,n] \bullet I_{\psi_l}^r [n,1]$. The entries in the $\alpha[1,n]$ matrix are scaling coefficients for the corresponding n number of $I_{\psi_l}^r$ values in the $I_{\psi_l}^r [n,1]$ matrix. The *dot product* (i.e. \bullet) yields the scaled TP_l^r values as shown by equation (3.3).

To summarize, the SDITPM framework, which is basically a PID controller, is defined by the following equations:

- a) (3.1) for planar integration: r specifies the region, K indicates the number of secondary metrics, and l uniquely identifies the primary metric.
- b) (3.2) for vertical integration: subscript S (e.g. in $TP_{r,S}$) indicates the stack-level effect.
- c) (3.3) for scaled control: the scaling matrix fine-tunes the $TP_{r,S}$ control.
- d) (3.4) for leveraging a single metric to yield TP_r : c identifies the migration decision cycle.
- e) (3.5) for leveraging multiple primary metrics to yield TP_{rS}^c .

$$I_{\psi_{l,K}}^r = \int_{j=1,r}^K (\psi_{l,r}^j) dj = \sum_{j=1}^K (\psi_{l,r}^j) \dots (3.1)$$

$$TP_{r,S} = \int_{l=1,r}^{STK} \int_{j=1,r}^K (\psi_{l,r}^j) dj = \sum_{l=1}^{STK} \sum_{j=1,r}^K (\psi_{l,r}^j) \dots (3.2)$$

$$TP_{r,S} = TP_{l,r} = [\alpha_r^{l=1}, \dots, \alpha_r^{l=Sub}] \bullet I_{\psi_{l,d}} \begin{bmatrix} I_{\psi_{l,1}} \\ I_{\psi_{l,2}} \end{bmatrix} \dots (3.3)$$

$$TP_O^c = TP_r^c \dots (3.4) \quad c = 1,2,3,\dots,Z \text{ (leveraging one primary metric in } c^{th} \text{ cycle)}$$

$$TP_O^c = TP_{r,S}^c \dots (3.5), \quad c = 1,2,3,\dots,Z \text{ (leveraging multiple primary metrics)}$$

The example in Table 3.1 demonstrates how equation (3.3) leverages three primary metrics. Each primary metric X contributes two secondary metrics, dX/dt for D control and M_i/M_{i-1} for P control. M_i denotes the current mean of the X variable at the i^{th} cycle, but his cycle may not coincide with the c^{th} cycle in equations (3.4) and (3.5); $i^{th} = c^{th}$. The scaled vertical integration is materialized by the equation, $TP_{l,r} = [\alpha_{l=1}^r, \alpha_{l=2}^r] \bullet I_{\psi_{l,r}} \begin{bmatrix} I_{\psi_{1,r}} \\ I_{\psi_{2,r}} \end{bmatrix}$ or its alternative $\alpha_{l=1}^r * I_{\psi_{1,r}} + \alpha_{l=2}^r * I_{\psi_{2,r}}$. The scaling coefficient α_i^r (e.g. $\alpha_{l=2}^r$ above) may be different for different control regions r .

In this example the number of deviation errors in control region r of the control plane indexed by l is $K = 2$; the deviation error is $\psi_{l,r}^j$ for $j = d = 1,2$ in the c^{th} transfer policy decision making cycle	$K_1 = \psi_{l,1}$ (deviation error for the 1 st secondary metric)	$K_2 = \psi_{l,2}$ (deviation error for the 2 nd secondary metric)	$I_{\psi_{l,j}}^r = \sum_{j=1}^K (\psi_{l,r}^j) \dots (3.1)$ is planar integration for region r
1 st primary metric (i.e. $l = 1$)	0.6	0.4	$I_{\psi_1} = I_{\psi_{l,j,r}} = 1$ (from equation (3.1))
2 nd primary metric (i.e. $l = 2$)	0.123	0.27	$I_{\psi_2} = 0.393$ (from equation (3.1))

3 rd primary metric (i.e. $l=3$)	0.82	0.37	$I_{\psi_3} = 1.19$ (from equation (3.1))
$TP_{r,S}^c = \int_{l=1,r}^{STK=3} I_{\psi_l} dl$ (equation (3.2))			$TP_{l=[1,3],r} = 1 + 0.393 + 1.19 = 2.583$
$TP_{r,S}^c$ is now scaled by equation (3.3), yielding $[\alpha_{l=1}^r, \alpha_{l=2}^r, \alpha_{l=3}^r] = [15, 1, 0.5]$			$TP_{l=[1,3],r} = 1 * 1.5 + 0.393 * 1 + 1.19 * 0.5$ i.e. $TP_{r,S}^c = TP_{l=[1,3],r} = 1.938$

Table 3.1. Three primary metrics (STK=3) and for computing $TP_{l,r}$; $TP_{rS}^c = TP_{l,r}$

The example in Table 3.1 explains the usage of equations: (3.1), (3.2), and (3.3). The region r in the example leverages three primary metrics (i.e. $l = 3$) that each contributes deviation errors computed from the two corresponding secondary metrics; that is $K = 2$. The regional transfer probability by vertical integration is given by the stack sum $TP_{rS}^c = TP_{l=[1,3],r} = 1 + 0.393 + 1.19 = 2.583$. After scaling it become 1.938. If this value is the highest regional transfer probability, then $TP_o^c = TP_r^c$ holds. Migration is affirmative for the condition $TP_o > Th_o$, where Th_o is threshold of the dominant currently control region where TP_o came from.

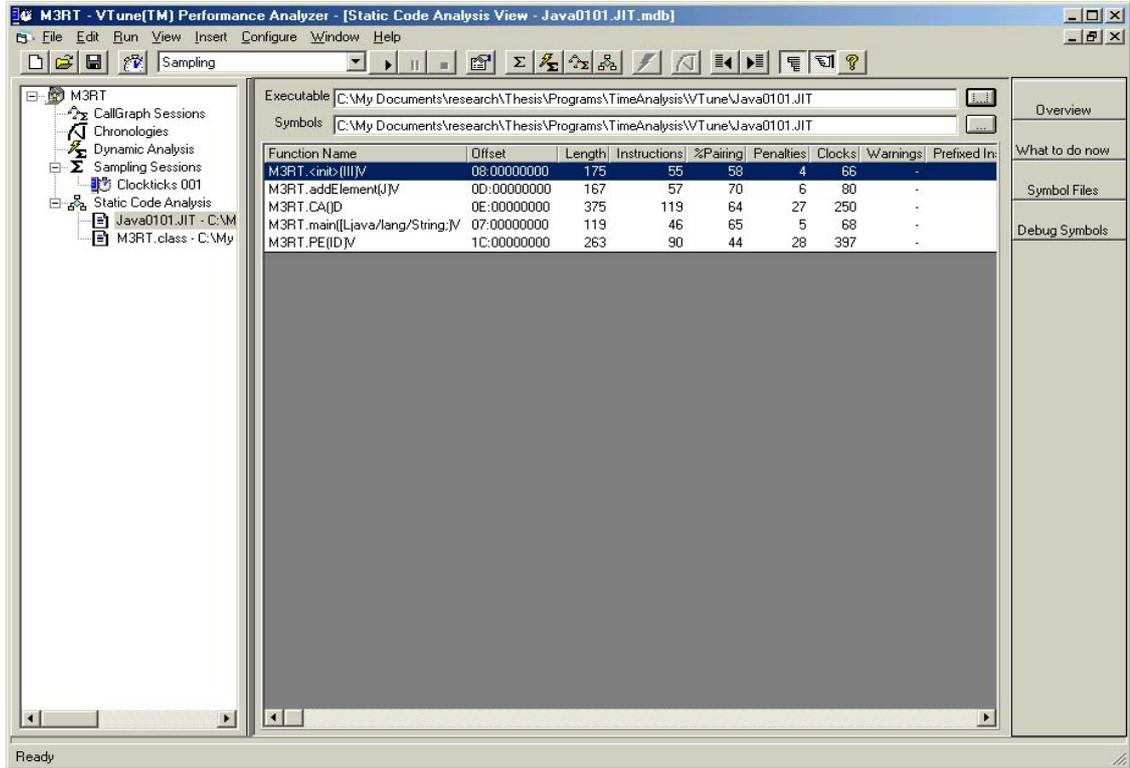


Figure 3.3. VTune timing analysis of the Java MCA implementation (i.e. M^3RT)

In the SDITPM framework the real-time computation of the mean, M_i of the primary metric at i^{th} cycle is by the M^3RT or MCA technique. The migration decision cycle c may not be in synchrony with the i^{th} M^3RT cycle (i.e. $c^{th} \neq i^{th}$). The P control is based on the $\frac{M_i}{M_{i-1}}$ ratio, but the D control can be obtained by one of the following two methods:

- a) *Using M_i directly:* The D control is $D_i = \frac{(M_i - M_{i-1})}{dt}$. It is more stable than the other method but less responsive because of the M_{i-1} feedback in the i^{th} cycle of M_i prediction.

- b) *Using simple linear regression (SLR):* This alternative method, which is executed in the i^{th} cycle either as $y_i = b_0 + b_1x_i + e_i$ or in the c^{th} cycle as $y_c = b_0 + b_1x_c + e_c$, is based on the *least-squares error* concept (e_i or e_c is the curve-fitting error) [Jain91]. It makes the D control responsive because it has no feedback at all. The parameter b_i is the rate of change in the i^{th} M_i cycle, and the x_i variables are the $(F - 1)$ number of data samples. The F value is the same as the *flush limit* chosen for the MCA implementation.

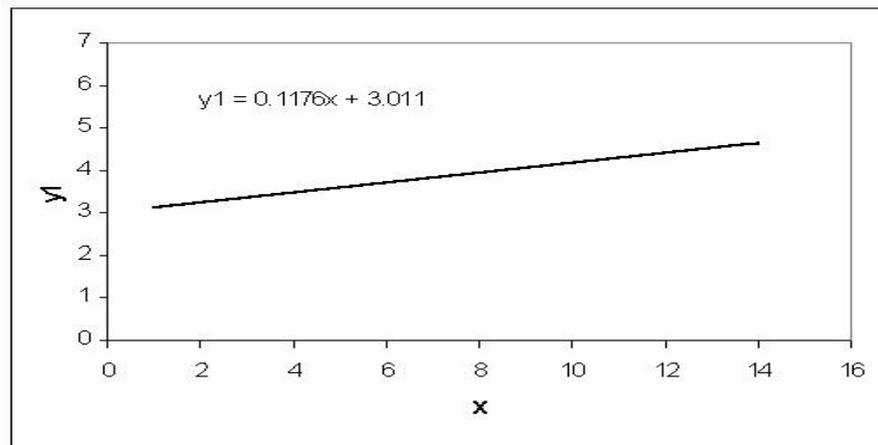


Figure 3.4. Simple linear regression of 13 queue length samples

The MCA and the *simple linear regression* (SLR) methods can be used for the D control implementation in the SDITPM framework because a) they are statistical and independent of the waveform type and b) they are simple and need only short computation times. The SLR needs simple computations for b_0 and b_1 from the data samples. The execution times for MCA and SLR are compatible and

they can be measured with the *Intel's VTune Performance Analyzer* [VTune] in number of clock cycles. The different *VTune* measurements confirm that the M^3RT object requires only an average of 330 clock cycles for intrinsic execution. Intrinsic means that data is immediately usable from a pre-collected trace. In field applications the actual M^3RT execution time is the sum of its intrinsic computation time and the total IAT (inter-arrival time) delays for collecting the $(F-1)$ number of data items. For a platform operating at 850 MHz the physical intrinsic M^3RT execution time is $[\frac{1}{(850 * 10^6)}] * 330 \approx 3.88 * 10^{-7}$ seconds. The short execution times required by M^3RT and *SLR* make them immensely suitable for stringent time-critical applications [Stankovic98]. Figure 3.3 is a screen capture for one of the *VTune* analysis the M^3RT intrinsic execution time. The *SLR* reflects the trend of change from the current data samples without referring to the past performance. This makes *SLR* vulnerable to noise and perturbations. Figure 3.4 is the simple linear regression for $(F - 1) = 13$ number of queue length samples on the X axis. The plot shows that the trend of the queue length in this case grows at the rate of $b_1 = 0.1176$ requests per unit time.

3.3 Connective Summary

This chapter describes the proposed SDITPM framework in details. Conceptually it can be represented by Figure 3.2, and formally it is represented by the set of equations, (3.1), (3.2), (3.3), (.4) and (3.5). Two waveform independent techniques are adopted for leveraging real-time statistics of the chosen primary metrics, *MCA* (*micro Convergence Algorithm*) and *SLR* (*simple linear regression*).

The timing analysis shows that both of them require more or less the same execution time. The more stable MCA yields a less responsive result. In contrast, the more responsive SLR approach is more vulnerable to system perturbations because it has no feedback loop similar that of the MCA approach. In the next chapter some verification results for the SDITPM prototype will be presented.

CHAPTER 4. VERIFICATION EXPERIMENTS

The SDITPM was verified in the Aglets environment in two ways: static and dynamic. The choice of the Aglets Mobile Agent Platform [AGLETS] is intentional because a) it is designed for the Internet and this makes the verification results scalable so that they can be validated in the open Internet, and b) the Aglets has rich user experience to draw from, and c) previous work indicates that the Aglets is stable. In the static way only a single computer is used, and a single resident aglet (agile applet) runs on it. This aglet samples the injected waveforms (e.g. Poisson, self-similar, or actual traces), which simulate the metric profiles, to compute the overall transfer probability for making migration decisions. In fact, the aglet, which embeds the SDITPM mechanism, does not migrate physically but the decisions provide the correspondence/correlation between the migration decisions and the current statistics. The user can visualize what changes that the static object sees in the process of time. If the migration decisions correspond with the injected statistics correctly, then the SDITPM mechanism is logically verified. In the dynamic way the SDITPM mechanism in the aglet, which simulates a migrating logical server, actually samples the chosen metrics in the network nodes to make migration decisions. The aim is to verify the following: a) the migration decisions indeed correspond with the current actual statistics of the metrics being leveraged, and b) the effect of load balancing is indeed achieved because of the object mobility.

Figure 4.1 summarize the two verification approaches. The static approach is marked as “Path 1” and the dynamic verification is called “Path 2”, which may

involve remote nodes that interact over TCP channels. The use of traces in the SDITPM verification exercise is an important step because these traces represent actual Internet operations that follow the power laws [Medina00]. To recap, static verification involves only one static/resident aglet and one node (host of the former) and therefore the simulations do not involve actual aglet/object mobility. In contrast, dynamic verification involves one mobile aglet that can migrate to any nodes within the predefined network boundary.

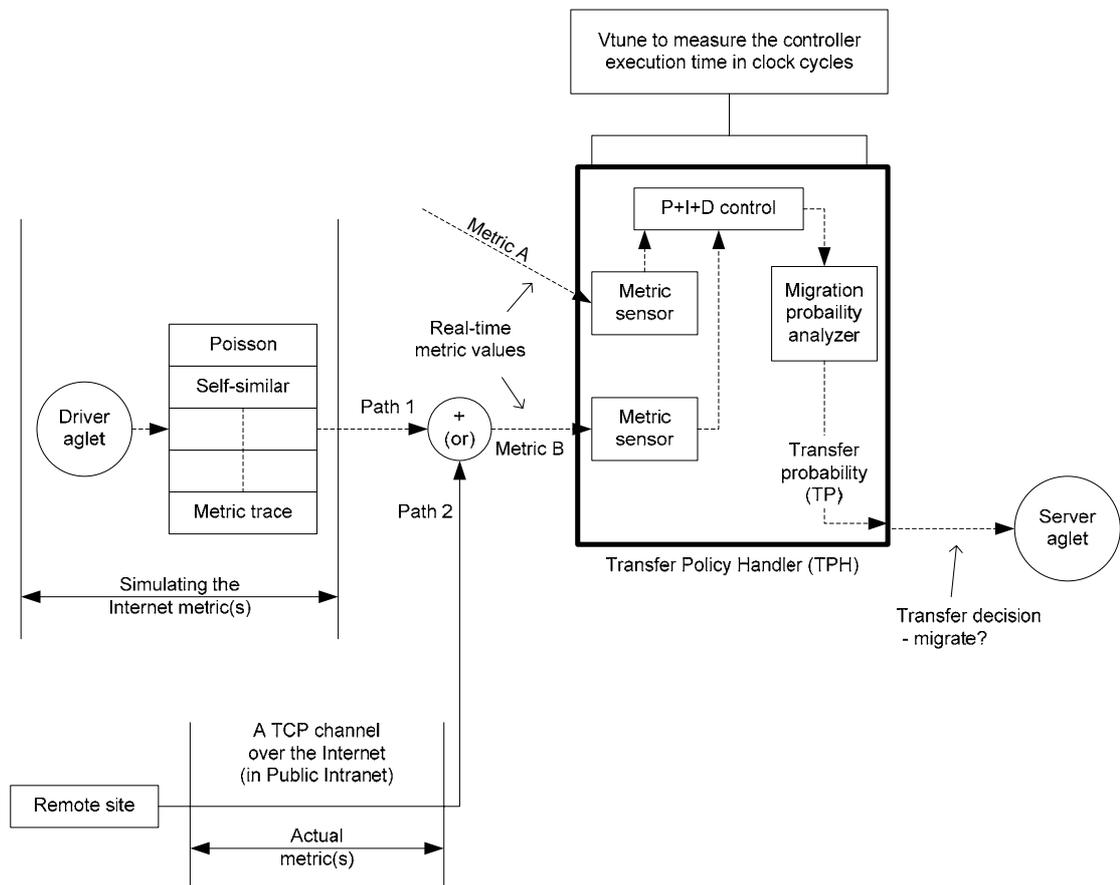


Figure 4.1. The Aglets environment for the SDITPM verification experiments

In the verification experiments the SDITPM prototype, which becomes a part of the logical server construct, is called the *Transfer Policy Handler* (TPH). As shown in Figure 4.1 it has the following elements:

a) *Metric sensors*: Each sensor is responsible for sampling a particular primary metric waveform/distribution. The number of sensors depends on how many metrics are leveraged.

b) “*P+I+D*” *control*: Every metric waveform is leveraged for purposes of P, I, and D control purposes. Transfer policy decisions are made based on the combined effect of the regional control values (Table 4.1). For example, the C_3^1 region (synonymously known as the following: [C1, C3], region R1, or PID control) leverages both the P and D controls in terms of the deviation errors: C1: $\psi_l^{C1} = P_l - (ref_P + \Delta_P)$ and C3: $\psi_l^{C3} = D_l - (ref_D + \Delta_D)$ respectively. The I control leverages the successive deviation errors from the secondary metrics for the predefined window K, namely,

$$I_{\psi_{l,d}} = \int_{j=1}^K (\psi_{l,d}^j) dj = \sum_{j=1}^K (\psi_{l,d}^j) \text{ (i.e. equation (3.1))}. \text{ The regions } C_3^2 \text{ and}$$

C_4^1 leverage only the D and P controls respectively, but the I control is present for all the regions. To be exact, the control mechanisms for the different regions in Figure 3.1 are as follows: a) “P+I+D” control for C_3^1 , b) “D+I” control for C_3^2 , and c) “P+I” control for C_4^1 .

c) *Migration probability analyzer (MPA)*: This module computes the transfer policy probability (or TP_r) for current regional control:

$TP_r = \sum_{l=1}^{Sub} [w_{l,r} * TP_{l,r}]$ (i.e. equation (3.3)). Then, the largest TP_r value

becomes the transfer probability TP_o of the overall control plane (Figure 3.1 and Figure 3.2). The primary metrics being leveraged are indicated by l , for $l = 1, 2, \dots, sl$. If the transfer decision cycle is indicated by c , then

$TP_o^c = TP_r^c = TP_r$ (i.e. equation (3.4)) holds for $c = 1, 2, \dots, Z$.

The transfer policy decision to migrate is affirmative for the $TP_o^c > Threshold_r$ condition, and the choice of $Threshold_r$ for the control region r (e.g. R1) is predefined and depends on the types of primary metrics being leveraged. At the present stage the $Threshold_r$ choice is left as an implementation issue.

D_1 control ↓ P_1 control →	C1 (P+)	C2 (P-)
C3 (D+) ($Th_{C_3^1}$, $Th_{C_4^1}$ and $Th_{C_3^2}$ are the predefined thresholds for the different regions)	<i>Migrate</i> for sure if only a single primary metric is leveraged by setting $TP_{r=C_3^1}$ to infinity so that $TP_{r=C_3^1} \gg Th_{C_3^1}$ always holds (region R1 or C_3^1 or [C1,C3] or Migration ; PID or “P+I+D” control)	<i>Alarm 1 (A1)</i> : for $TP_l^{C_4^1} < Th_{C_3^2}$, migrate otherwise (region R2 or C_3^2 or [C2,C3] or Alarm 1; DI or “D+I” control)
C4 (D-)	<i>Alarm 2 (A2)</i> : for $TP_l^{C_4^1} < Th_{C_4^1}$, otherwise migrate (region R3 or C_4^1 or [C1,C4] or Alarm 2; PI or “P+I” control)	<i>Don't care or Inert</i> state (no action) (region R4 or C_3^2 or [C2,C4] or Inert; inert control state)

Table 4.1. SDITPM decision matrix for a control plane (one metric)

The synonyms for the planner control regions as shown in Table 4.1 which is a more detailed description of Figure 3.2, are as follows:

- a) Control region 1: Can be referred to as R1, C_3^1 or [C1,C3] for PID control.
- b) Control region 2: Can be referred to as R2, C_3^2 or [C2,C3] for DI control.
- c) Control region 3: Can be referred to as R3, C_4^1 or [C1,C4] for PI control.
- d) Control region 4: Can be referred to as R4, C_3^2 or [C2,C4], an inert control state.

4.1 Static Verifications

The aim of the static verification experiments is to ensure that the SDITPM mechanism is logically correct. Even though the aglet that embeds the SDITPM mechanism does not physical migrate, the experimental data can empirically indicate the following: a) the migration decisions indeed correspond to the sampled statistics of the metrics being leveraged, and b) the SDITPM decisions work for different numbers of metrics being leveraged at the same time. The following sections demonstrate migration decisions versus different number of metrics.

4.1.1 Single Metric

The SDITPM can leverage any metric and treats it simply as a waveform. The following experiments for demonstration purposes leveraged different pre-collected traces, which were injected as waveforms by the driver shown in Figure 4.1 to excite the SDITPM mechanism. These traces included the CPU utilization profiles, the memory utilization profiles, and the context switching time profiles. Different examples will be presented in the sequel for demonstration purposes.

4.1.1.1 Memory Utilization

In this experiment the trace of system memory consumption or utilization profile was used. The profile is normalized to a range of 0-1 (1 for 100%) as shown in Figure 4.2. The '0' value means that all system memory resources are free, whereas '1' represents no more free memory.

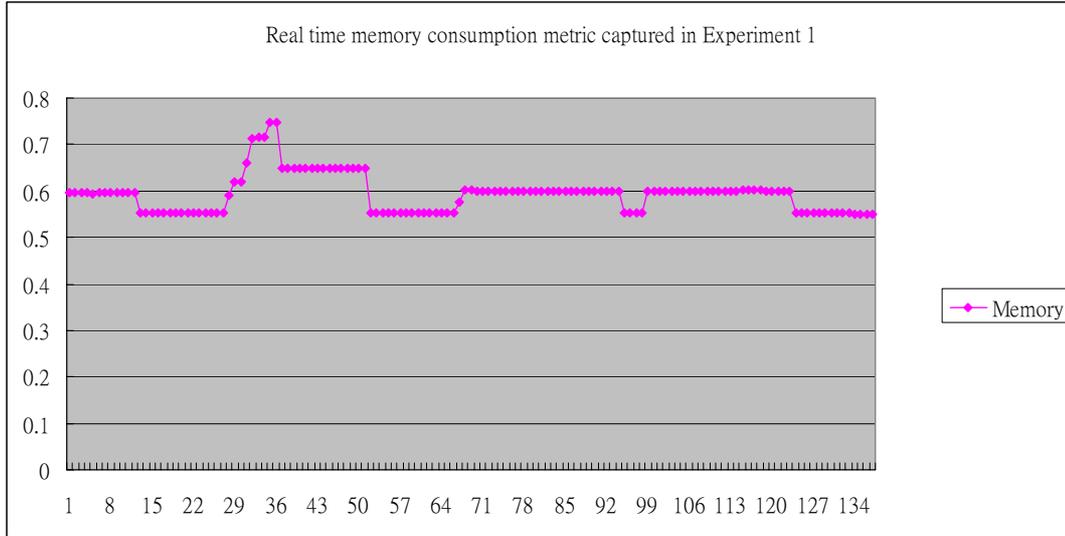


Figure 4.2. The memory utilization profile/trace

The SDITPM mechanism in the aglet (logical server) leveraged the profile for making migration decisions. The decision is affirmed if the condition the “self” overall transfer probability TP_o is greater than the given threshold. The TP_o is the collective result computed from the regional transfer probabilities TP_r . The concept of regional thresholds Th_r and the migration decisions are plotted in Figure 4.3. P control in the SDITPM mechanism is M_i/M_{i-1} , where M_i is computed by the Convergence Algorithm (CA) in its micro form called the M^3RT [Lo02, Wong01]. The rate dQ/dt or D control is measured by the difference, $M_i - M_{i-1}$. Table 4.2 is an example that illustrates how M_i/M_{i-1} and dQ/dt can be calculated with respect to the i^{th} prediction cycle of the Convergence Algorithm (CA), which is part of the

SDITPM construct. Every CA prediction cycle uses thirteen data samples to calculate the current mean M_i of the waveform being leveraged.

i^{th} prediction cycle	1	2	3	4	5	6	7	8
M_i	0.553	0.591	0.690	0.620	0.659	0.713	0.580	0.717
M_i/M_{i-1}	-	0.936	0.857	1.113	0.941	0.924	1.229	0.809
dQ/dt	-	0.038	0.099	- 0.070	0.039	0.054	- 0.133	0.137

Table 4.2. Real-time calculations of M_i/M_{i-1} and dQ/dt

The SDITPM has four control regions, Migration (R1), Alarm 1 (R2), Alarm 2 (R3) and Inert (R4). The deviation error in a time window is calculated as follows:

1. When both P and D controls fall outside their tolerance bands the deviation error for region R1 will be $\Psi_p + \Psi_d$, whereas the deviation errors for the regions R2, R3, R4 are set to zero. Ψ_p is the deviation error (with respect to the row in Table 4.1) for proportional or P control, and Ψ_d is the deviation error (with respect to the column in Table 4.1) for derivative or D control.
2. When only the D control falls outside its tolerance band the deviation error for R2 will be Ψ_d , whereas the deviation errors for R1, R3, R4 are set to zero.
3. When only the P control falls outside its tolerance band the deviation error for R3 is Ψ_p , and the deviation errors for R1, R2, R4 are set to zero.

4. When the P and D controls are inside their tolerance bands there is no migration decision necessary. This is why R4 is an inert control state.

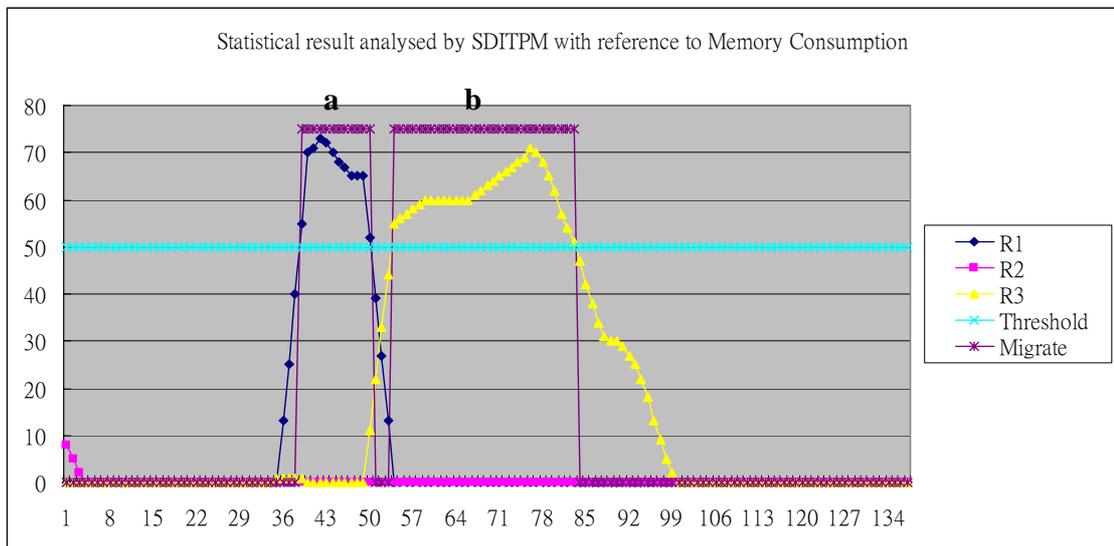


Figure 4.3. SDITPM migration decisions for the memory utilization in Figure 4.2

Figure 4.3 shows the values of the three different regions (i.e. R1, R2, and R3) over time. For simplicity all the control regions have the same threshold value (i.e. $Th_1 = Th_2 = Th_3 = 50$). The rising edges of rectangular pulses indicate when migration decisions were made. For example, the rising edge of the rectangular pulse “a” is the point when the logical server would have migrated if it were not static. The migration decision was affirmative because the PID control value in R1 exceeded the given threshold. That is, the dominant control value for “a” is the PID mechanism. Similarly; the rising edge the rectangular pulse “b” indicates PI control, which became dominant instead. Since the aglet server is static/resident it sees all the changes of dominance from one control region to another. The experimental result shows that SDITPM can indeed make sound transfer policy decisions for

server migration even by leveraging a single metric. Similar experiments unanimously identified that the memory utilization profile is a good metric for SDITPM.

4.1.1.2 CPU Utilization

Figure 4.4 is the CPU utilization profile used in the experiment. The migration decisions made by the SDITPM mechanism are shown in Figure 4.5. The rising edges of the three rectangular pulses indicate when server migration decisions were affirmed. The three decisions were all dictated by the R1 control (i.e. CPU utilization is too high (P control) and increases too steeply (D control)). Since the aglet server is resident it sees all the changes of dominance from one control region to another. For example, within the “a” pulse width the R1 (i.e. PID) dominance was later replaced by R3 (i.e. PI). This experiment further confirms that the SDITPM mechanism has no problem making sound transfer policy decisions for object/server migrations by leveraging a well chosen metric such as the CPU utilization. In fact, similar experiments unanimously identified that the CPU utilization profile is a useful metric for the SDITPM to make sound migration decisions.

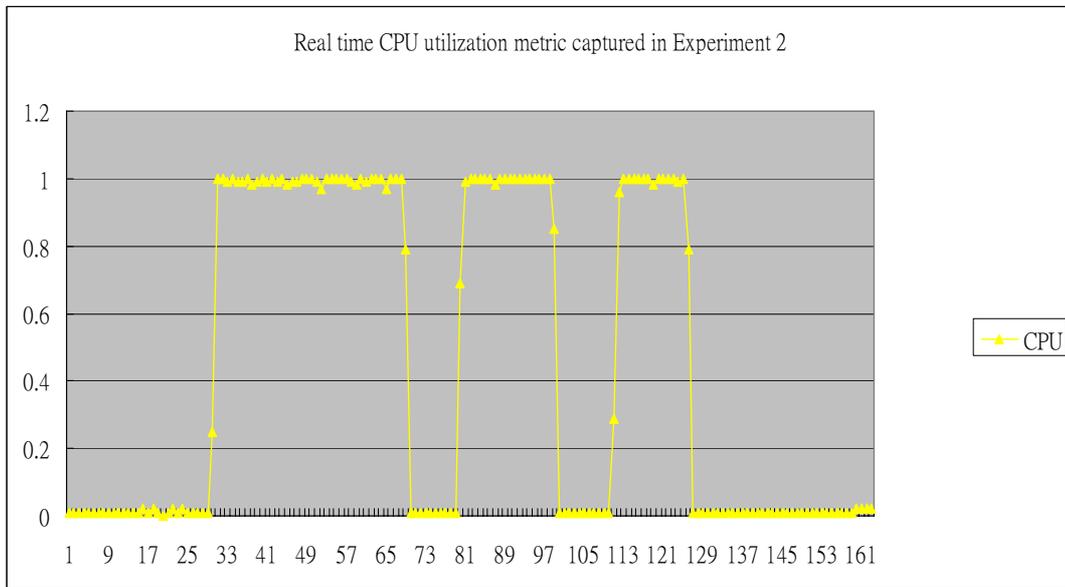


Figure 4.4. A CPU utilization profile

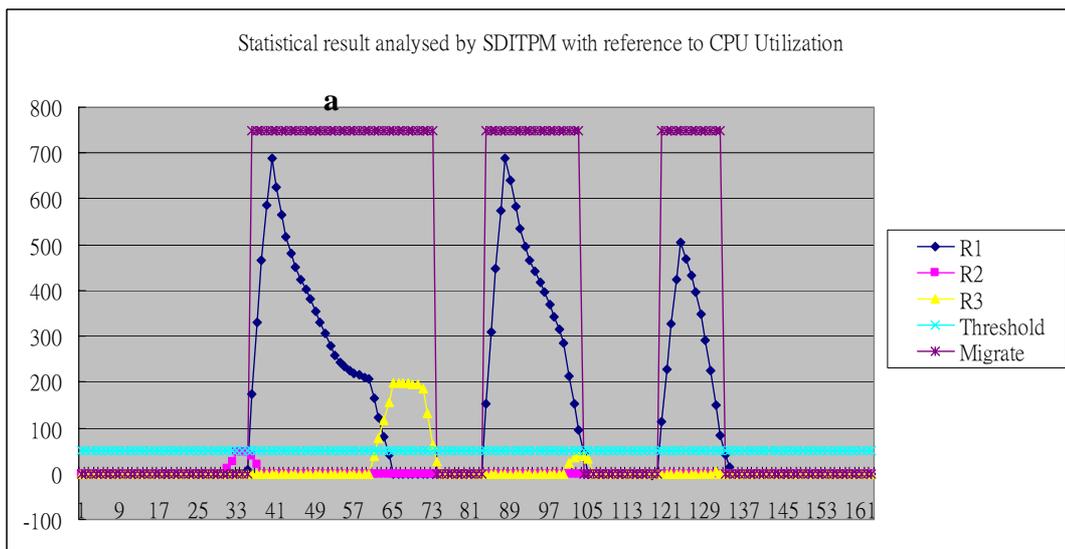


Figure 4.5. Migration decisions by SDITPM based on a CPU utilization trace

4.1.1.3 Context Switching Time

Figure 4.6 is the CST (context switching time) profile being leveraged by the SDITPM mechanism in the resident aglet. Figure 4.7 shows that the first and second

migration decisions were based on the transfer probability of R1 (i.e. PID), but the third migration decision is due to R3 (i.e. PI control).

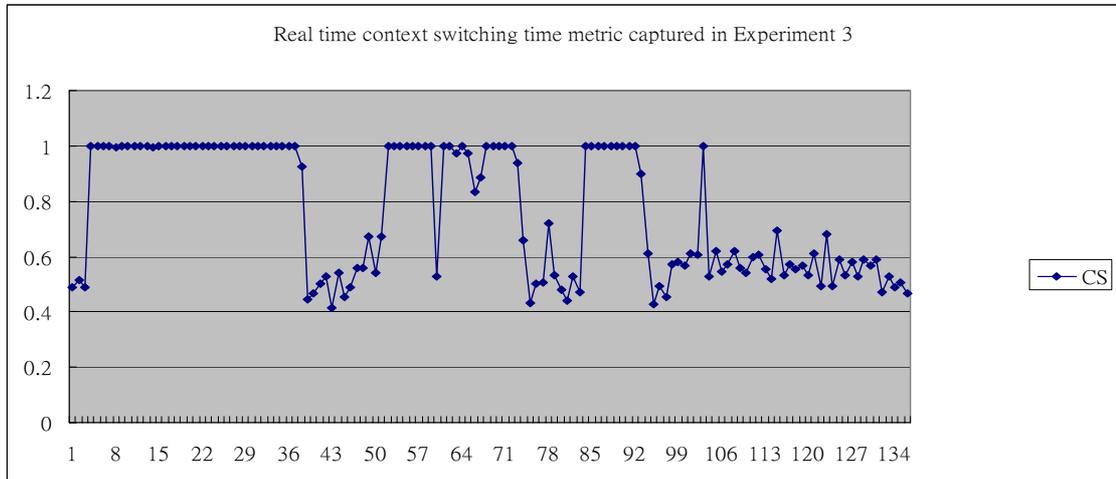


Figure 4.6. Context switching time (CST) profile/trace

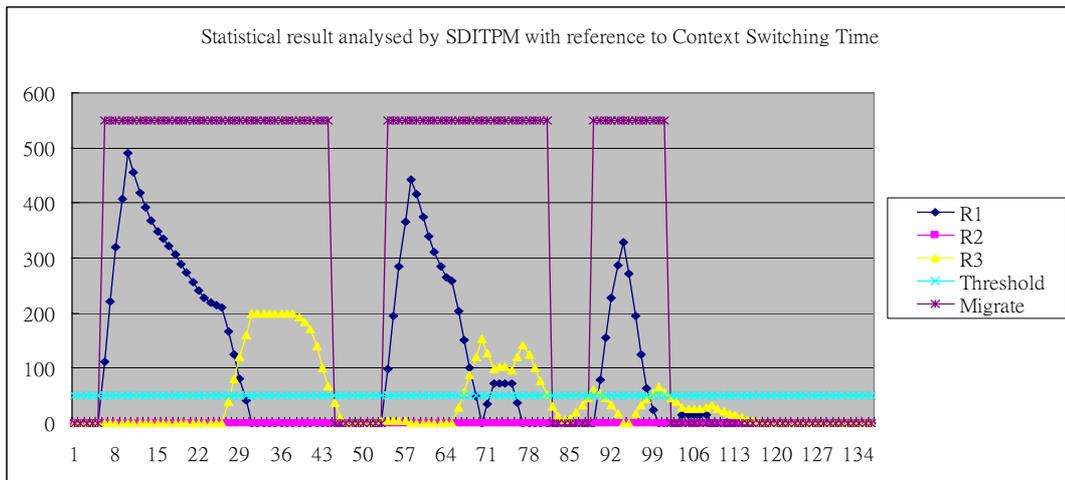


Figure 4.7. Migration decisions for the CST trace in Figure 4.6.

4.1.2 Leveraging Two Metrics

Figure 4.8 shows two traces, memory and utilizations together. When these two metrics were leverage by the SDITPM mechanism at the same time the deviation errors of the metrics were summed. The summation is the basis of the

planar integral control shown in Figure 3.2. Therefore, the overall transfer probability TP_o should theoretically be larger compared to single-metric leveraging.

The corresponding migration decisions are shown in Figure 4.9.

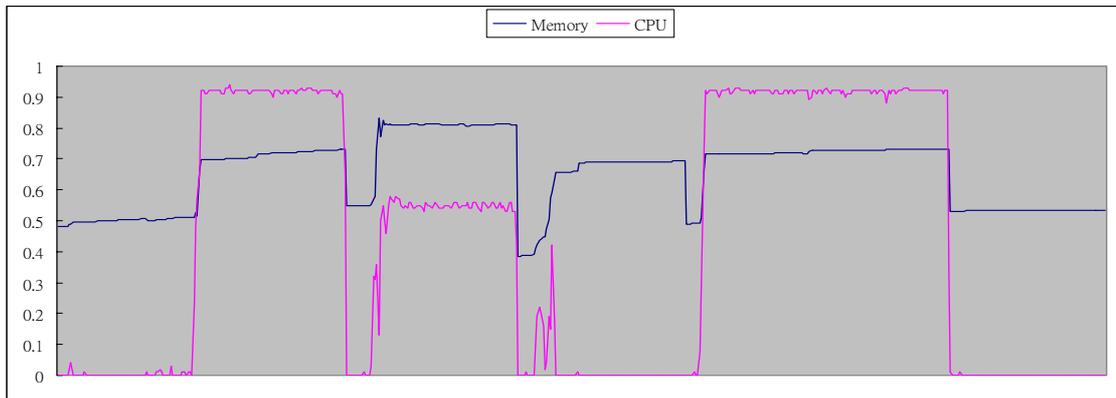


Figure 4.8. Two superimposed metrics, memory and CPU utilizations

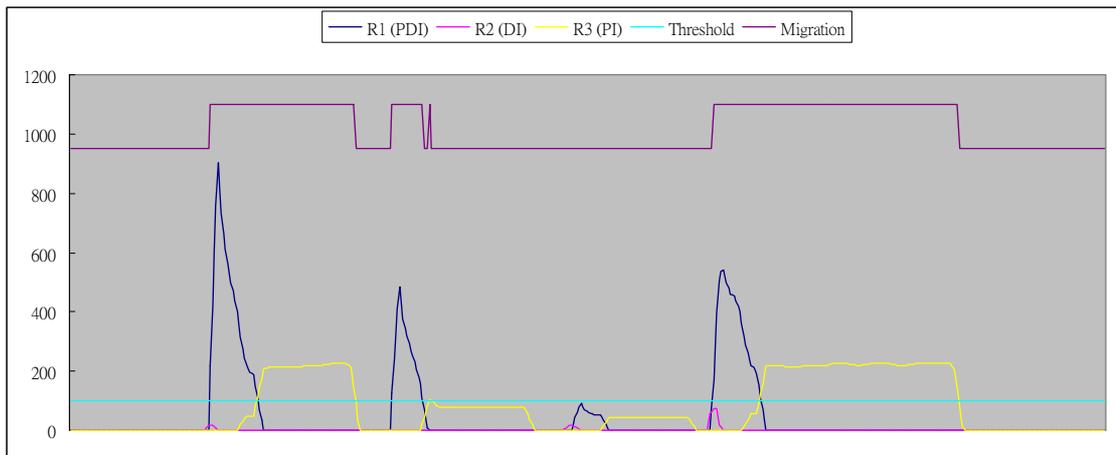


Figure 4.9. Migrations by leveraging memory and CPU utilizations simultaneously

4.1.3 Leveraging Three Metrics

In this experiment three traces, memory utilization profile, CPU utilization profile, and the logical context switching time were leveraged. Figure 4.10 shows the three metrics, and the migration decisions are shown in Figure 4.11 and Figure

4.12 of different thresholds. The result shows that SDITPM has no problem in leveraging multiple metrics for making migration decisions. Figure 4.11 and 4.12 indicate that the SDITPM control sensitivity can be tuned by setting different threshold values. For example, there are four migration decisions in Figure 4.11 with threshold equal to 50 but five decisions in Figure 4.12 with threshold equal to 180.

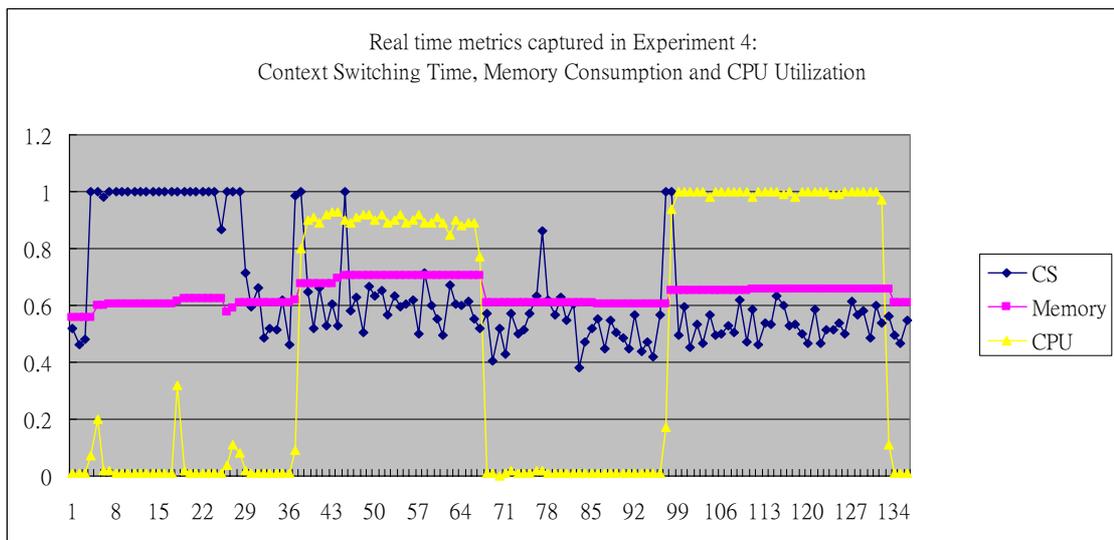


Figure 4.10. Superimposed context switching time, memory and CPU utilizations

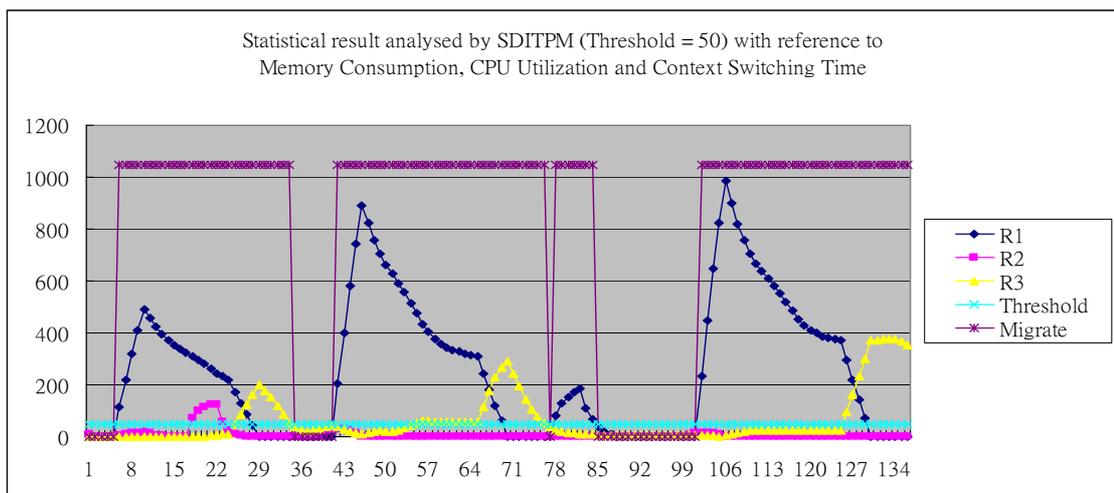


Figure 4.11. Migrations by leveraging metrics shown in Figure 4.10 (threshold = 50)

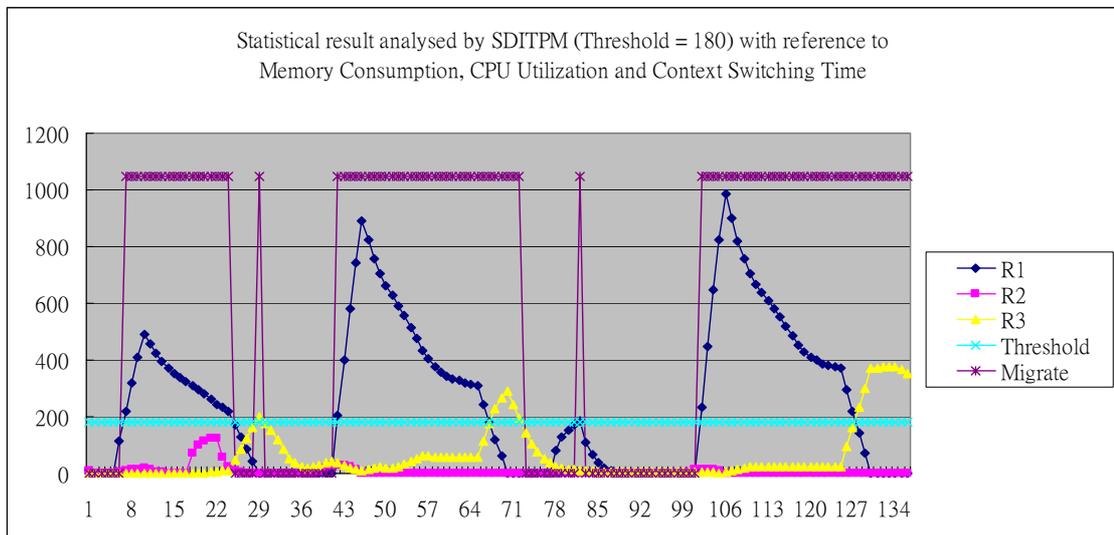


Figure 4.12. Migrations by leveraging the metrics in Figure 4.1 (threshold = 180)

4.2 Dynamic Metric Leveraging

Dynamic leveraging of metrics by the SDITPM mechanism for making migration decisions needs the support of a network architecture called the Public Intranet. Since there are many details involved, the whole of Chapter 5 will be dedicated to the discussion of this issue. In order to provide a glimpse of what does it mean by metric leveraging at this stage CPU utilization (U) is used as an example. The U value is a powerful metric as shown in some of the experimental results in this thesis. The SDITPM framework leverages it as a primary metric in a real-time manner in light of the following features:

- a) It is a distribution of utilization (e.g. in %) versus time.
- b) It provides two useful secondary metrics for the SDITPM,

- i) Proportional control, $P = U_i / U_{i-1}$,
 - ii) Derivative control, $D = dU/dt = (U_i - U_{i-1}) / (t_i - t_{i-1})$
- c) The safety margins Δ in the P and D objective functions $\{0, \Delta_P\}^2$ $\{0, \Delta_D\}^2$ respectively divide the control plane into 4 control regions (Figure 3.1), and each is governed by the regional threshold. The integral (I) control for each control region at any time sums up the current deviation P and D errors, which are absolute values greater/smaller than the safety margin (i.e. Δ_P and Δ_D).
- d) If the integral control value is larger than the regional threshold, the chance of object migration is affirmed.

4.3 Connective Summary

The novel SDITPM is a framework for real-time migration decision making. It was verified in two ways, static and dynamic. Static verification means that one aglet, which embeds the SDITPM mechanism, does not migrate physically. This aglet is excited by different kind of waveforms, including pre-collected traces, to simulate different metrics. The SDITPM samples the excitations and makes migration decisions. The SDITPM is considered logically correct provided that the decisions correspond to the sampled statistics. In the dynamic verification exercise the aglet, which emulates the migrating logical server that embeds the SDITPM mechanism, should migrate correctly and effectively within the predefined network boundary. The load balancing effect due to its object mobility should be visible if the SDITPM works as expected. The visibility can be confirmed from the

performance data that the aglet's self transfer probability TP_o is consistently the lowest compared with the intrinsic ones. The rationale is that with the SDITPM support the logical server always migrates to the least busy villager node. Since the dynamic verification exercise involves a lot of details, the whole of Chapter 5 is dedicated to its discussion.

CHAPTER 5. DYNAMIC SDITPM VERIFICATION AND VISUALIZATION OF OBJECT MOBILITY

Dynamic SDITPM verification involves a single logical entity that migrates on the fly to different nodes within the boundary of a subnet carved out of the Internet (e.g. Intranet). It is a known fact it is not easy to debug distributed programs that run on distributed hardware. Traditional techniques are not suitable for debugging distributed software because distributed objects have their own address spaces. Instead, visualization techniques [Wong2001A3, Wong2001Words] are lauded as powerful candidates for effective distributed software debugging. There were many publications since the early 1990s trying to address the problem of debugging distributed software (e.g. Myers90, Herrate91 and Yeung98b). However, there is little progress in this area as indicated by the more recent mobile agent platforms, which usually lack a efficacious visualization package for deciphering abnormal object behaviour (e.g. Aglets, Ara in Table 5.1) The aim is to record and decipher abnormal behaviour for the collaborating cognate objects. This is easier said than done because there is a network boundary problem to overcome. Firstly, it is impossible for any agent to migrate anywhere in the open Internet at will without any prior collaboration agreement among “villages” or “farms” of nodes [Wong00]. Secondly, even when there is an agreement there should be a mechanism in a village/farm to record all the agent activities in a designated shared-memory area, which could be centralized or distributed. Even the relatively more well-known existing mobile agent platforms such, as Ara [Ara], Aglets [Aglets] and Concordia (Table 5.1), lack a powerful visualization support for debugging abnormal agent behaviour. The graphical interfaces of the visualization tools of Aglets and

Concordia do allow the user to record object locations but do not provide any means for the user to identify and decipher abnormal object behaviour. For this reason the novel generic A^3 framework was proposed in [Wong00] to support:

- a) Formation of a commune called the Public Intranet (PI) to facilitate mobile agent based computing, independent of the mobile agent platform of language.
- b) VUI (Visual User Interface) for visualizing object mobility.

For example, the VUI described in [Wong00] was adapted to support the Aglets mobile agent platform. By computing the migration index on the fly through leveraging the chosen system metrics in a real-time manner, for example, an aglet can migrate freely within the PI. The typically useful system metrics include the CPU utilization of the host and its memory utilization. If the migration index computed by the aglet has exceeded the given threshold, then migration could materialize depending on whether a suitable destination could be found.

	IBM Aglets	Concordia	Ara
Developer	IBM Tokyo Lab.	Mitsubishi Electric's Horizon Systems Laboratory	U. of Kaiserslautern
Programming language	Java	Java	C,C++,Tel
Code migration	Archived Object class, jar (compressed object code) file	Object Class	Object Class
Interface for Visualizing and debugging object behaviour	Tahiti: for managing agents and logging their locations only	For minimum remote object administration	None

Table 5.1. Comparison of some Agent Systems

The results produced by repeating the experiments described in [Wong00] indicate that the A^3 framework implemented in the Aglets environment indeed can

support effective visualization of object mobility. For this reason the A^3 framework was adapted to support the static and dynamic verifications of the SDITPM model. The details of these verifications will be provided later. The dynamic verification exercise was carried out in the distributed Aglets environment; the logical server and the evaluators (to be described later) were aglets. The experimental results in this chapter differ from the previous ones, which were obtained from simulations by using a single computer with the Aglets setup for static SDITPM verification.

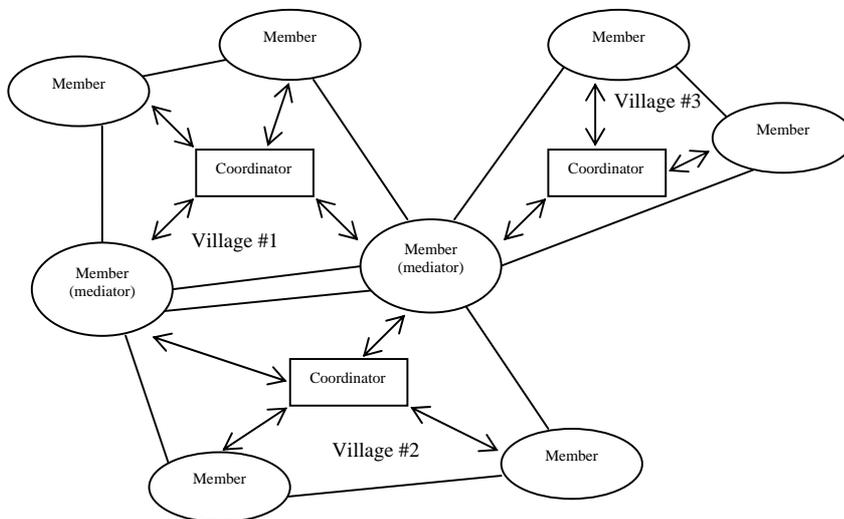


Figure 5.1. Three villages interact through the mediators to form a Public Intranet (PI)

5.1 Some Adapted A^3 Framework Details

The *adapted A^3 framework* (i.e. A^4 framework) is the visualization tool for the dynamic SDITPM verification experiments. The adaptation includes the following new features proposed as part of my MPhil thesis: a) real-time displays of the “self” regional transfer probabilities of the migrating logical server/aglet, b) real-time displays of the “intrinsic” regional transfer probabilities of the resident evaluator of the “console”, c) collecting the real-time profiles of the leveraged

metrics, and d) logging the migration decisions by SDITPM. In the A^4 context a “console” is the node where the VUI (i.e. MUI in the light of the console) is invoked.

The original features of the A^3 framework (still in A^4) are: a) coordinator declarations, b) village member registrations/deregistrations, c) generic VUI interface of two parts: the *manual* part that allows PI creation and the *visualizer* part that shows the current PI configuration and traces the object mobility. In the A^3/A^4 context, villages are interconnected in a non-adhesive manner by the mediators or connectors to form a large commune called the Public Intranet (PI) (Figure 5.1). Any Internet node however distant can join and leave a village at will by membership registration/deregistration. Only commune members can share the communal resources and non-members are regarded as intruders. A mediator is an Internet node that has memberships in different villages. Through its multiple memberships it has become the gateway for agents to migrate from one village to another to share the local resources with the mediator’s privileges by referral. The formation of a commune starts with the following three basic steps:

- a) Any Internet node becomes the coordinator of a village by declaring itself. The declaration is achieved by executing the public “*Coordinator.class*” program (CP). A successful execution means inclusion of the new coordinator in the public Coordinator File (CF). If a coordinator wants to give up its role, it should inform its villagers to deregister and executes the CP again to terminate its role. The VUI is a generic API to be invoked from any Internet node called the console (VUI for a particular console is the MUI).

- b) Any Internet node can become a member of any village by first executing the “*Register.class*”. The execution brings out the VUI/MUI through which the user chooses to join the preferred village(s). This is achieved because the MUI displays the CF contents that show all the villages. This will be demonstrated by a walkthrough later.
- c) For a new member the local coordinator broadcasts/multicast its identity and address to other members. With the new information the villagers can update their local records of current communal members. These local records are called the Network Configuration Tables (NCT). Deregistration is simply a reverse process in which the deregistered member will be deleted from the NCT.

5.2 A Walkthrough of the Original A³ Framework

A³ stands for three issues of fault tolerance [Gartner99]:

- (a) *Access security*: The idea is that if agents do not come from the same commune, which is either a village or a public Intranet (to be explained later), they are denied access to the communal resources.
- (b) *Agent persistence*: The aim is to enable a mobile agent to move out of dangerous zones, ahead of possible partial failures, for self-preservation. Anticipations of possible faults are achieved by computing the reliability indices based on the chosen set of run-time parameters.
- (c) *Avoidance of long queuing time for service*: The issue is to let an agent move out of a site that requires long waiting (queuing time) for service. Since this type of agent migration from a busy site to an idle one would even out the workload for different sites, it is treated as dynamic load balancing in the A³ framework. This

migration is decided by another set of indices, which are also computed from the instantaneous values of chosen run-time parameters.

The components of the Public Intranet are the villages that are controlled and managed by the corresponding coordinators. Any node can become a member of different villages by registration. Conceptually any Internet node, however distant, can become a village coordinator if it executes the “*Coordinator.class*” program or CP. For the simulation experiments all the programs are written in Java. The identities of different village members are displayed in the Coordinator Interface (CI) as shown in Figure 5.2. A coordinator, however, cannot be a member of any other villages. The identities of all the coordinators are stored in a “*Coordinator File*”, and are displayed in the Member User Interface (MUI). In Figure 5.3, the MUI is the VUI dedicated to the console `atp://u5x-172:8001/` (or simply `u5x-172`). Figure 5.2 is the CI for the coordinator `atp://u5x-170/`, which has no village member yet at the moment.



Figure 5.2. CI for the coordinator - `atp://u5x-170:8001/`

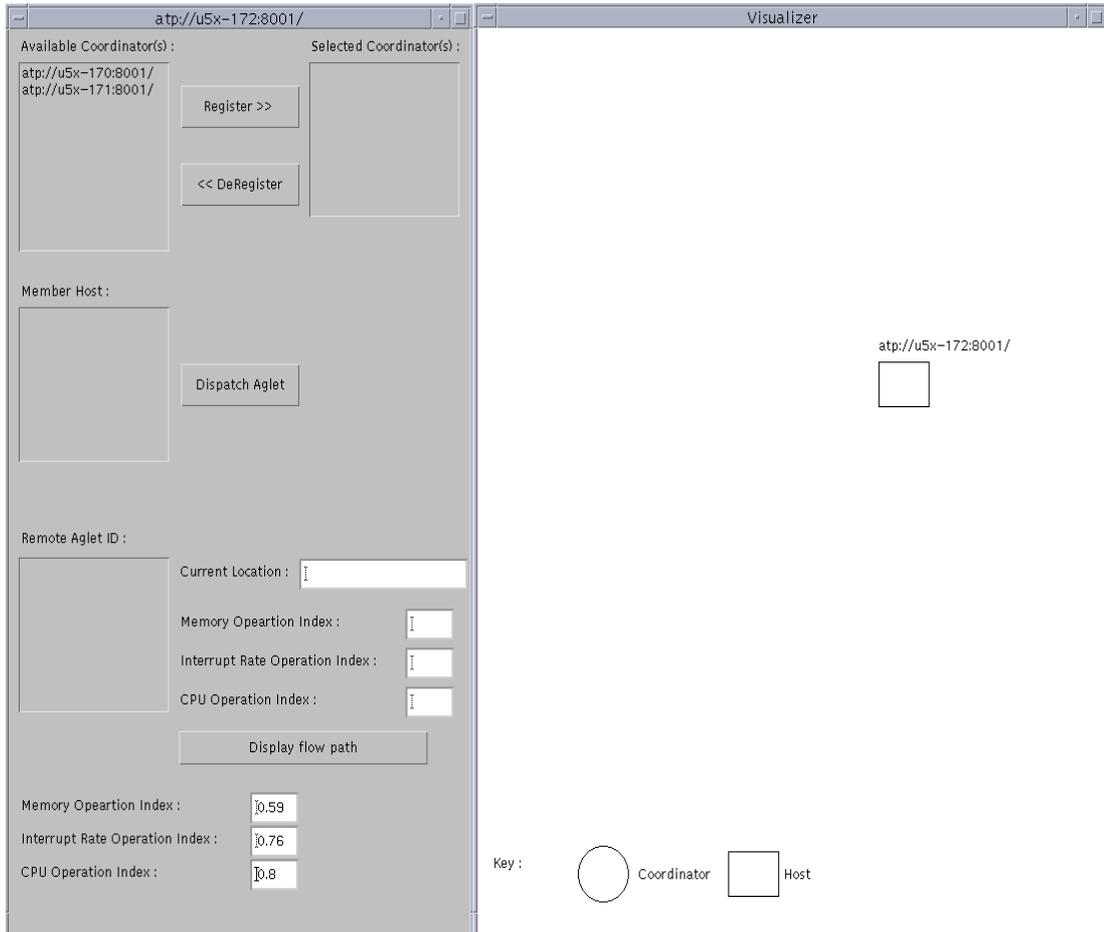


Figure 5.3. Member User Interface (MUI) at member node atp://u5x-172:8001/

5.2.1. Membership Registration

Any node in the Internet, other than the incumbent coordinators, can register as a member of any village by executing the “*Register.class*” Java program. If the program runs successfully, the local MUI (e.g. Figure 5.3) will appear and shows:

- a) The available coordinators in the network (in the Figure 5.3 case the

coordinators are: atp://u5x-170/ and atp://u5x-171/).

b) The user node identity (it is the Internet node atp://u5x-172/ in Figure 5.3).

An Internet node can register with more than one coordinator to become a mediator. The mediators serve as the connective that interconnects different villages to form a large commune or Public Intranet. Membership registration can be carried out if the user executes the public *Register.class* program. This brings out the MUI through which the user can choose and register its Internet node (i.e. the “console”) with any village(s) by highlighting the target node and clicking the “*Register>>*” button. The same process applies to deregistration except the button should be “*<<DeRegister*”.

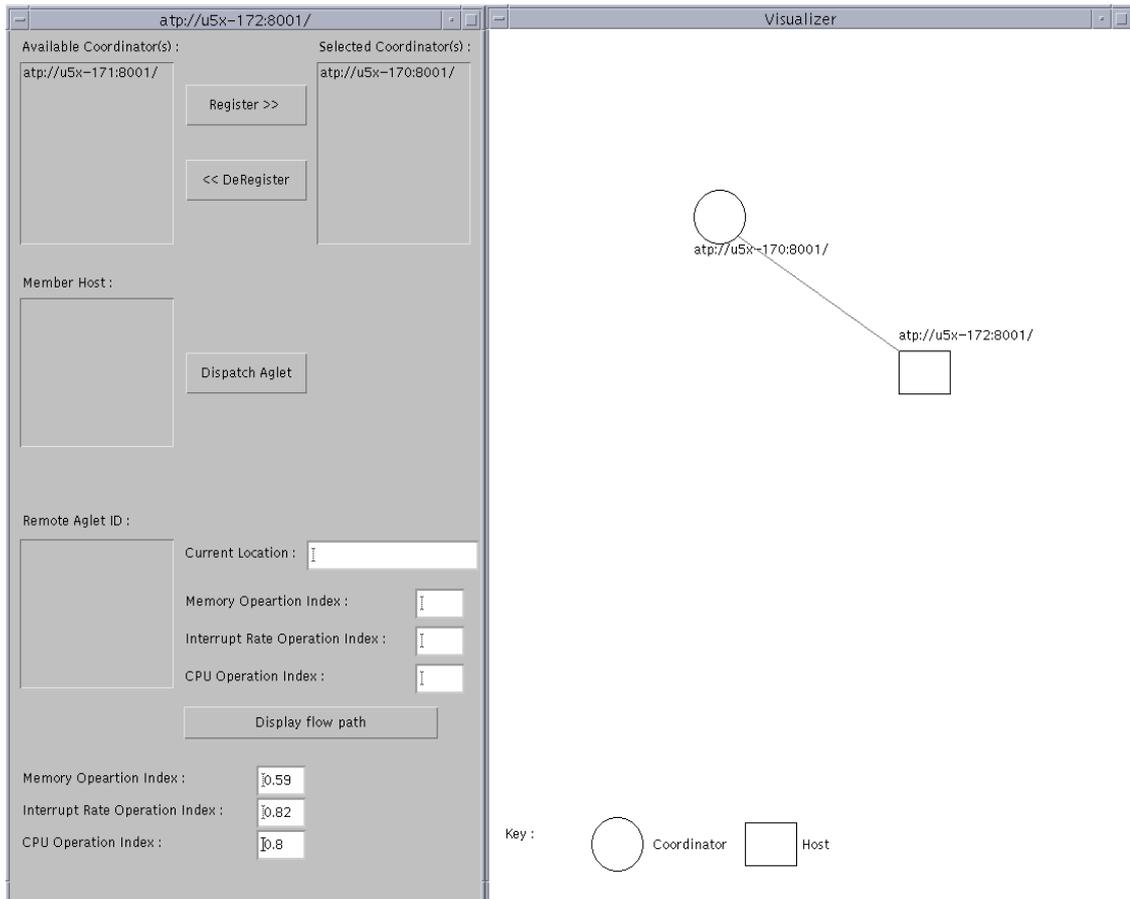


Figure 5.4 MUI for node (atp://u5x-172:8001/) shows the connection between atp://u5x-170 and atp://u5x-172 after a successful registration operation by atp://u5x-172



Figure 5.5. CI the atp://u5x-170) coordinator lists the current member

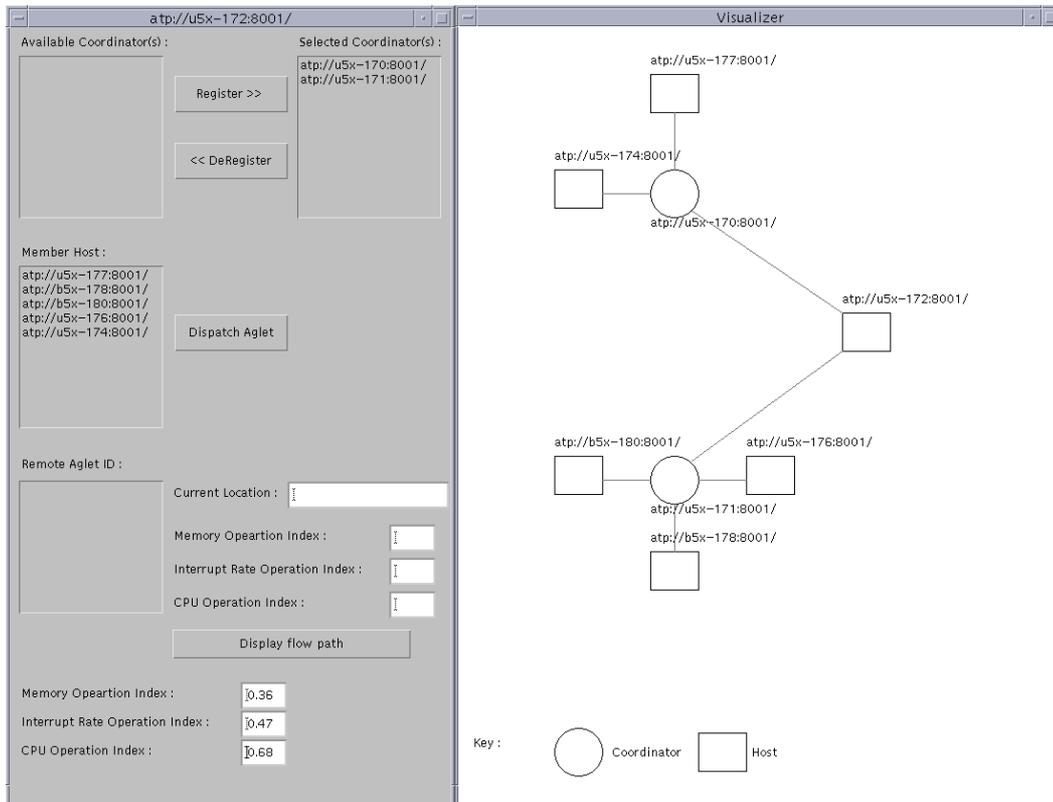


Figure 5.6. The mediator *atp://u5x-172:8001/* has joined two villages

To recap, a successful registration means:

- a) The node has become a village member and now has the right to share the resources from other members in the same village or other village(s) through the mediator.
- b) Its identity will be recorded in the coordinator's Host Table (Figure 5.5).
- c) It can not register with the same coordinator again.

The Figures, 5.4, 5.5, 5.6 and 5.7 walk through the sequence of how a village/PI is formed. The smallest PI is a single village. In Figure 5.4 the visualizer part of the MUI displays the connection between the villager (*atp://u5x-172*) and its coordinator explicitly (*atp://u5x-170*). The positions of the manual part and the visualizer part in a MUI can be swapped for the sake of presentation convenience and clarity. The

MUI in Figure 5.6 shows that the same Internet node cannot register with the same coordinator again for the “*Available Coordinator(s)*” field and now only has the remaining coordinator `atp://u5x-171`. In Figure 5.5 the Coordinator Interface (CI) shows its new villager `atp://u5x-170` in the Host Table. The visualizer part in Figure 5.6 captures the current PI, and the data in the manual part (left-hand side) matches this graphic PI representation (the right). The indices shown in the manual part (e.g. Memory Operation Index) characterizes the original VUI design [Wong00]. Figure 5.7 is another example that shows how the particular `atp://u5x-171` coordinator had managed four villagers at the time of the screen capture.

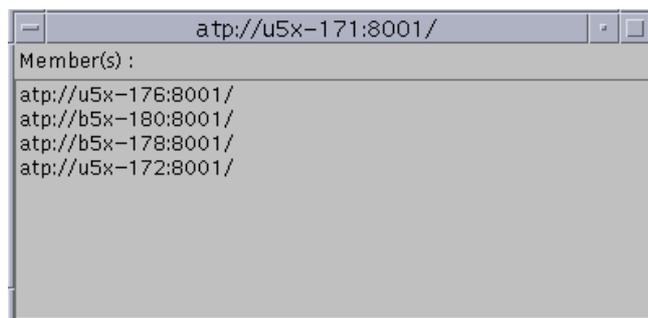


Figure 5.7. Node `atp://u1x-139:900/` is also a member in the village managed by coordinator `atp://ulx-137:9100/` as shown by its CI

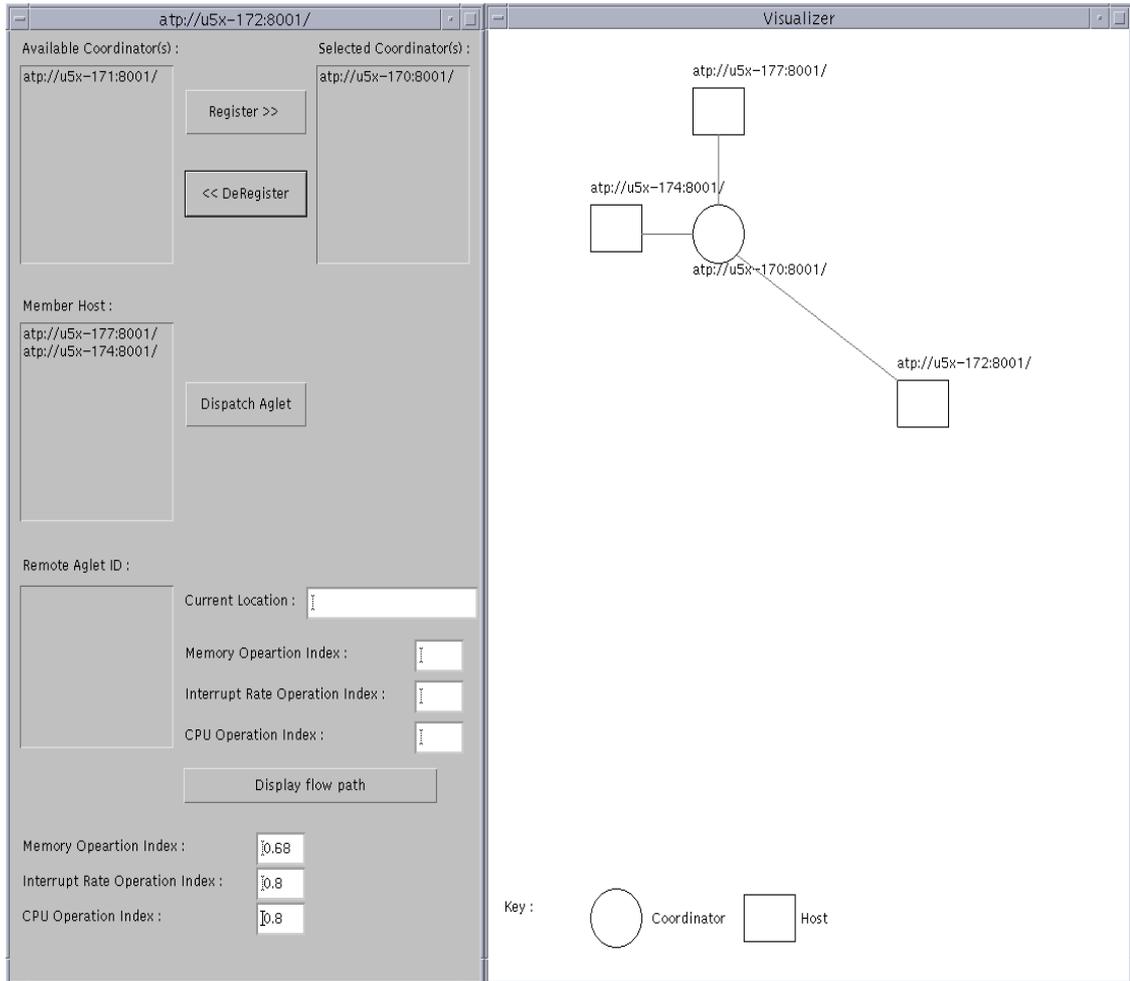


Figure 5.8. Node atp://u5x-172 has de-registered from atp://u5x-171

A node from any village is deregistered by clicking the “<<DeRegister” button in the local MUI. For example, the node *atp://u5x-172* is de-registered successfully from the village controlled by the *atp://u5x-171* coordinator, as shown in Figure 5.8. The result is disappearance of the whole village managed by the u5x-171 coordinator from the screen.

5.2.2 Tracing the Object Mobility

The mobility of an object can be traced by programming the object to migrate under certain conditions. Object migrations in [Wong00] were based on

different migrations indices (MI). The calculation of these indices differs completely from that for the SDITPM verification experiments. To differentiate the symbol P-MI (i.e. previous MI) denotes those calculations described in [Wong00]. When the P-MI values were greater than the given thresholds migrations were triggered.

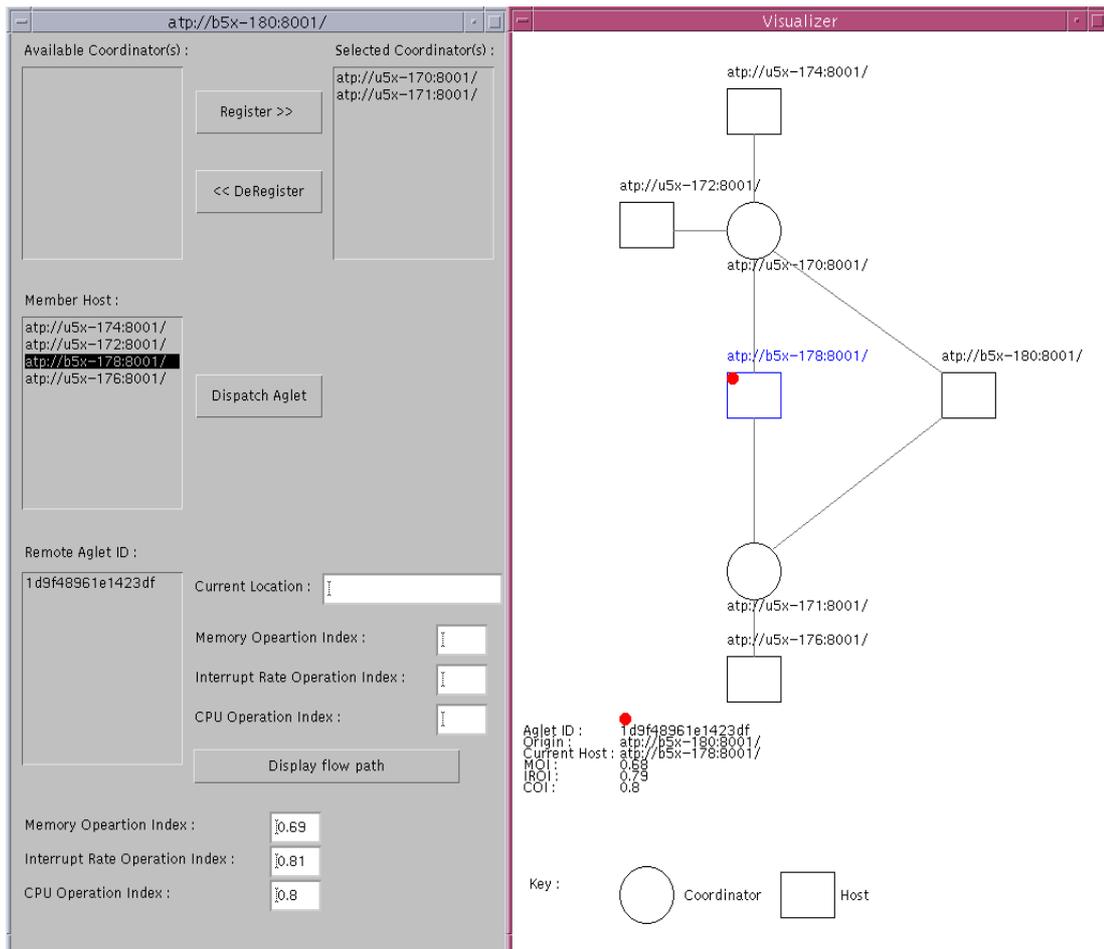


Figure 5.9. Object placed/dispatched to node atp://b5x-178

The migration and path-tracing processes in [Wong00] were manually achieved through the MUI by steps as follows:

Step 1. Click the destination node (e.g. the highlighted *atp://b5x-178* in Figure 5.9) to dispatch a mobile object (i.e. an aglet) from the *atp://b5x-180* console.

Step 2. Click the “Dispatch Aglet” button to actually dispatch the mobile object (now shown as a “dot” in *atp://b5x-178* of Figure 5.9).

Step 3. Capture the migration process, the mobile object had automatically migrated from *atp://b5x-178* to *atp://u5x-172* as captured by Figure 5.10.

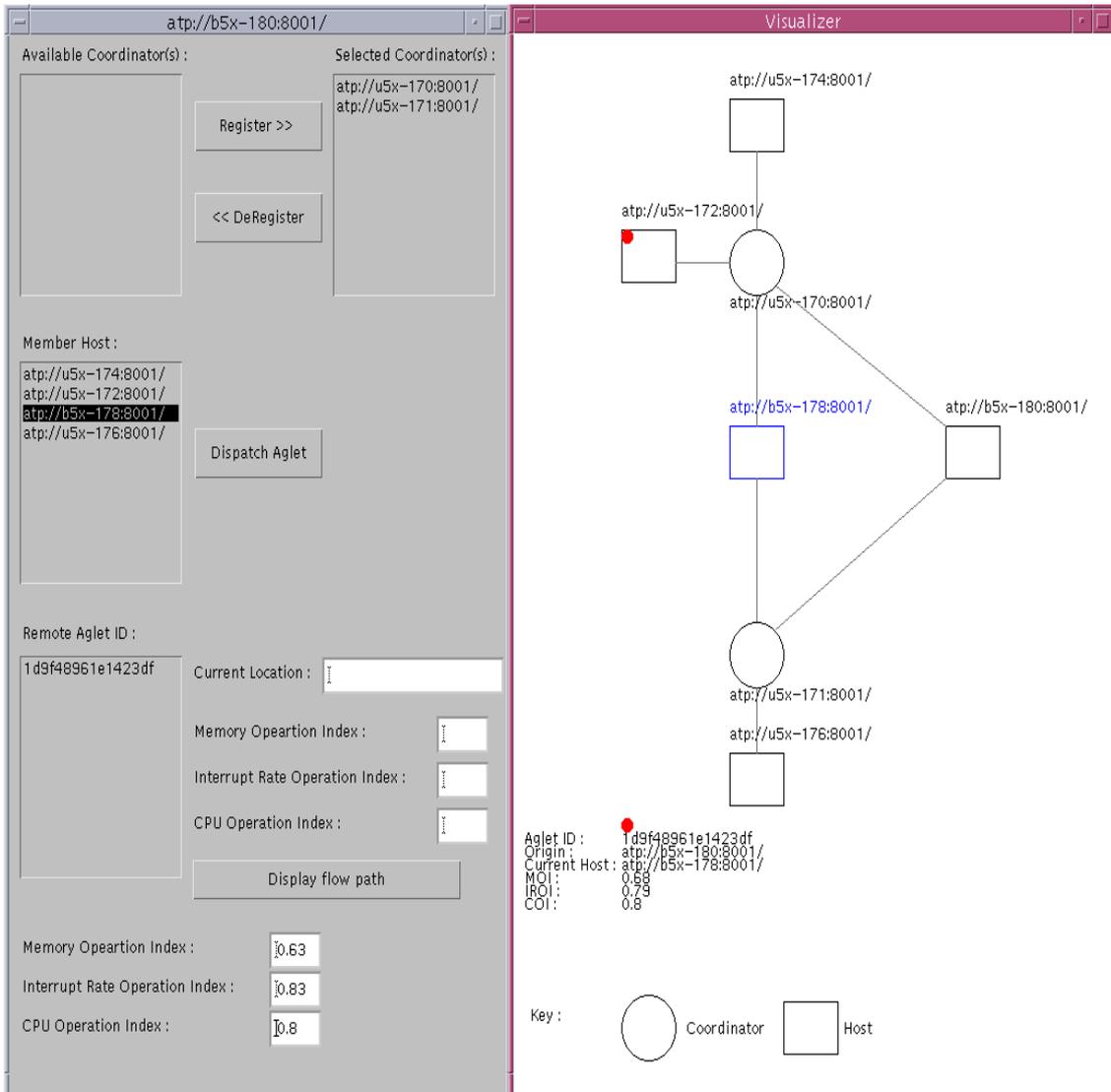


Figure 5.10. Object automatically migrated to node *atp://u5x-172*

5.3 Tracing SDITPM Object Mobility

The operation follows the philosophy described by [Wong00]. The original features in [Wong00] alone, however, cannot satisfy the need of the SDITPM verification exercise. Therefore, new features have to be added and they will be elaborated later. Object migration in the SDITPM verification exercise is not based on P-MI values but the overall transfer probabilities TP_o of the collaborating nodes in the PI. The local or intrinsic TP_o of a physical village node is computed by its resident evaluator entity, which is also an aglet. A migrating object, which is a logical server in the SDITPM context, leverages the same metrics from its host node as by the resident evaluator, but can be in a somewhat different way. Therefore, the TP_o computed by the logical server is called the “self” TP_o . For example, the evaluator may measure the average context switching time (CST) of the host node independent of how jobs are scheduled to time-share the CPU (i.e. intrinsic CST). In contrast, the logical server is interested only in its own CST (i.e. self CST), which can be very different from intrinsic CST. The difference depends on the execution priority of the logical server in the host. The self CST has a special meaning because it not only indicates the interval t since the logical server had virtually owned the CPU last time but also implies temporal locality. The meaning of temporal locality is that the chance P_t for the logical server to virtually own the CPU again is inversely proportional to t , $P_t \propto \frac{1}{t}$. The “self” and “intrinsic” properties lead to different regional control values computed by the SDITPM elements in the logical server and

the resident evaluator respectively, even the rest of the metrics being leveraged is the same.

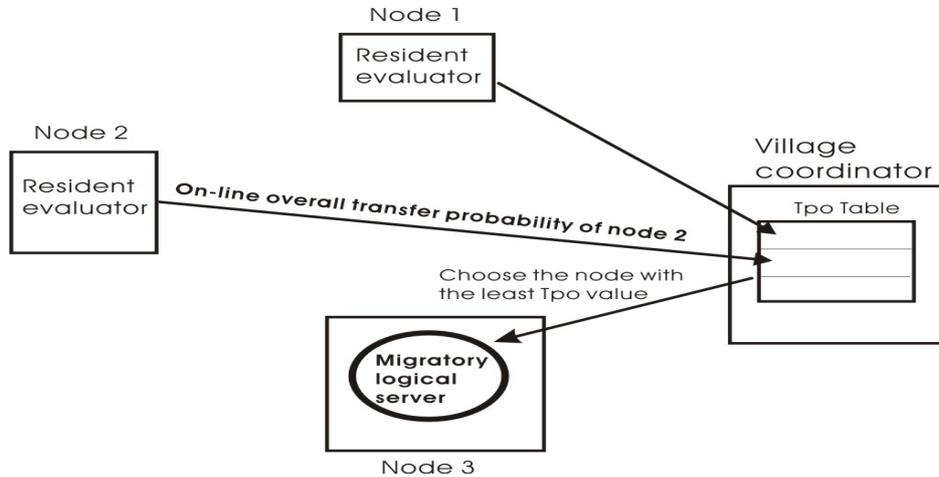


Figure 5.11. Resident evaluators and a coordinator's TP_o table

Each coordinator collects and updates all the local intrinsic TP_o values of the village members within the PI boundary kept in its TP_o table. It exchanges the TP_o information in this table with other coordinators in a dynamic manner. At the system steady state all the coordinators should have a complete picture of all the intrinsic TP_o values within the PI anytime. If a logical server needs to migrate because its self TP_o is larger than the given threshold, it picks the host with the lowest intrinsic TP_o in the coordinator's TP_o table as its target. In this way object mobility, which is represented by the migration path can be traced to verify the logical validity SDITPM framework. The essence of the local TP_o table kept by a coordinator is shown in Figure 5.11.

If the "Display flow path" button in the manual part of a MUI is pressed, the log of the migration path traced out by the logical server in the experiment will be

displayed. This enables the observer/user to decipher/debug any possible abnormal behavior in the light of object mobility. Figure 5.12 and 5.13 are two migration path examples logged by the A^3 and A^4 frameworks respectively.

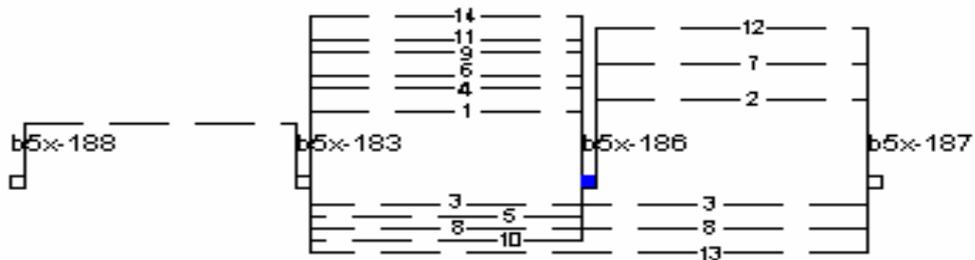


Figure 5.12. A migration path by A^3 framework without SDITPM support

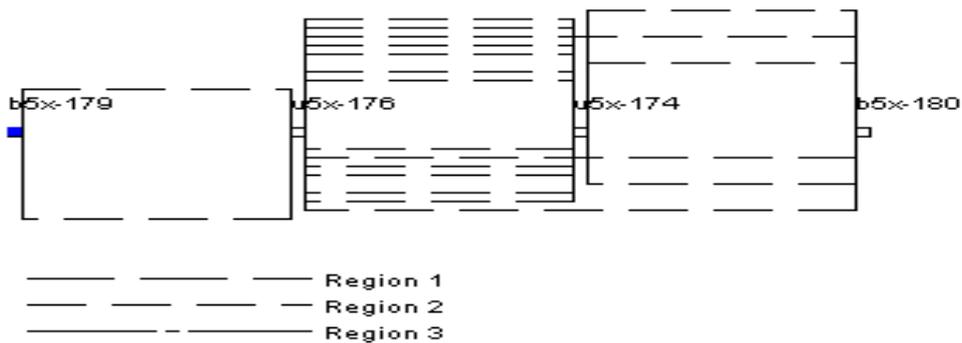


Figure 5.13. Migration path of a logical server supported by A^4 framework

5.4 The Distinctive A^4 Features

The A^4 framework is created from the A^3 version to facilitate the SDITPM verification exercise. By itself the A^4 framework is an original contribution because it is a novel tool for monitoring the SDITPM behaviour. The distinctive A^4 features (not in A^3) include the following:

1) *Real-time displays for the three SDITPM control regions*: The three control regions, namely, “Region 1”, Region 2”, and “Region 3” are displayed separately for both the local evaluator of the console and the logical server. The MUI in Figure 5.14 is for the console atp://b5x-180 (i.e. the syntax is atp://<host ip>:<port of the aglet>).

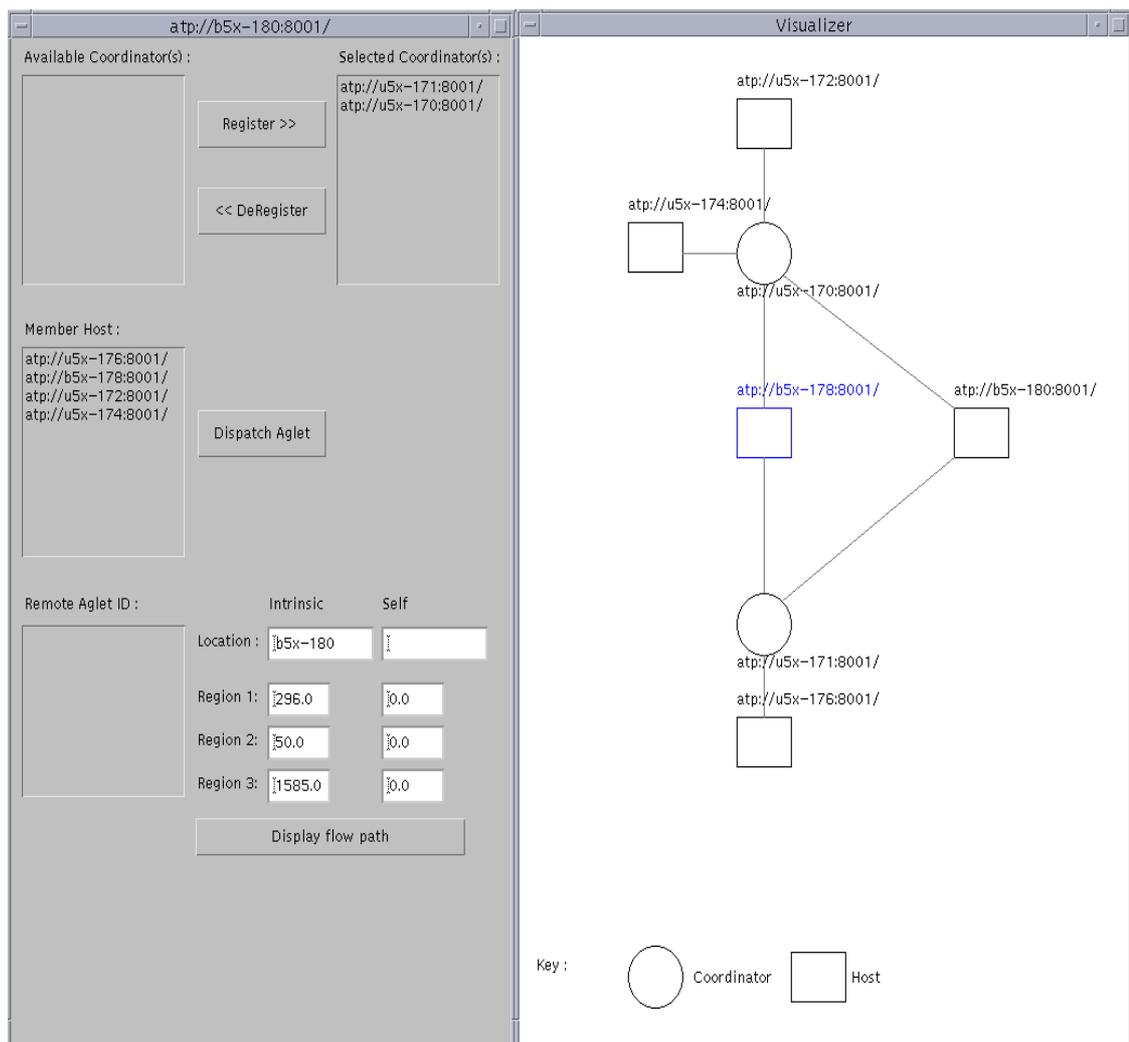


Figure 5.14 Display of the SDITPM control values by the b5x-180 console’s MUI

The intrinsic values of the control regions for the SDITPM mechanism of the resident evaluator reflect the current situations in the console. The self values for the

control regions of the SDITPM in the logical server are all zeros in this case because no logical server has been dispatched yet. Figure 5.15 displays the control values for both the resident evaluator and the logical server, which now runs in atp://u5x-176. The intrinsic regions show the control value for the console host, namely, stp://b5x-180. The self control values reflect the situations of the current host (i.e. atp://u5x-176) as far as the logical server is concerned.

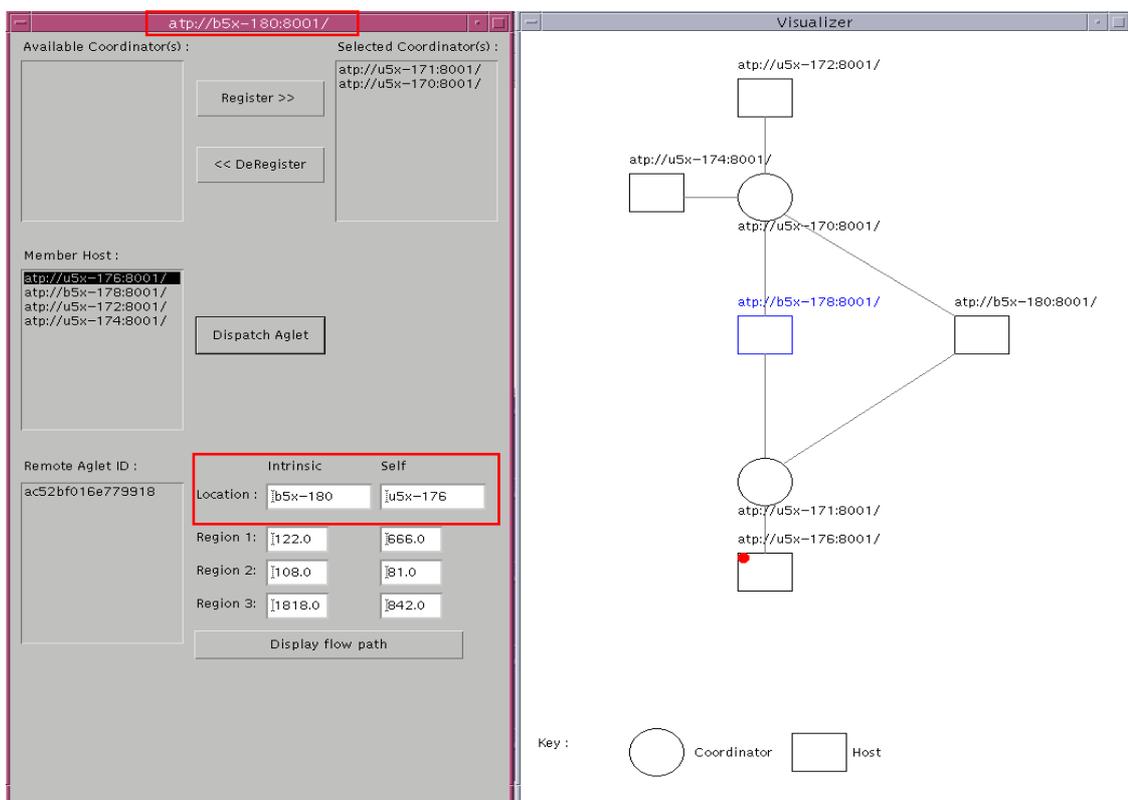


Figure 5.15. An example of collecting statistics after the logical server is dispatched

2) *The logging/terminal (L/T) screen:* There is a minimizing function (button) in the rightmost side at the top of the visualizer (e.g. Figure 5.15). The function is in the toggle state. If it is pressed, it will change to the logging/terminal screen as illustrated in Figure 5.16; one more press changes it back. The L/T screen has two parts. The upper part (i.e. the L window) logs when migration decisions were made

by the SDITPM transfer policy. Migration is triggered if the self TP_o is greater than the given threshold. This is shown as “MIGRATION is needed (TP > Th)” in Figure 5.17. In fact, the first migration decision was not materialized because there was no suitable target host. Finding the villager with the lowest TP_o as the target node for migration has achieved the effect of a location policy, which usually finds a suitable target by using cost functions. In the SDITPM verification exercise the TP_o values, in effect, represent the cost functions. In the Figure 5.16b case, the migration decision had actually materialized only after a few trials; the logical server eventually migrated to atp://b5x-178.

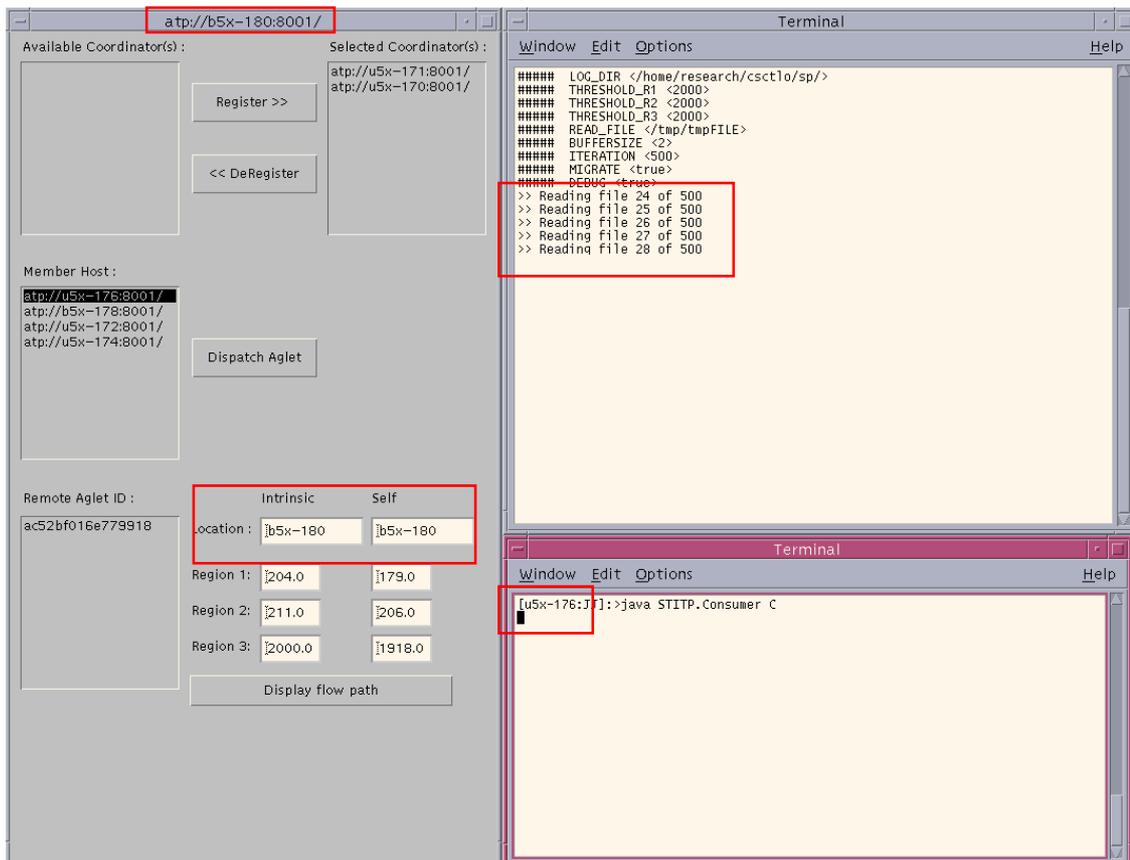


Figure 5.16 A L/T screen example

The T window is for creating statistics by injecting perturbations into the different metrics, for example, CPU utilization (or C), memory utilization (or M) and intrinsic context switching time (or P). For example, the command “[u5x-176:JJ]:>java STITP.Consumer C” in the T window in Figure 5.17 artificially injecting perturbations in the CPU utilization (i.e. C) in node atp://u5x-176. The injection is carried out by the STITP.Consumer class. In fact, the T window is terminal emulation that allows the user to inject perturbations to any remote node and retrieve its logged statistics.

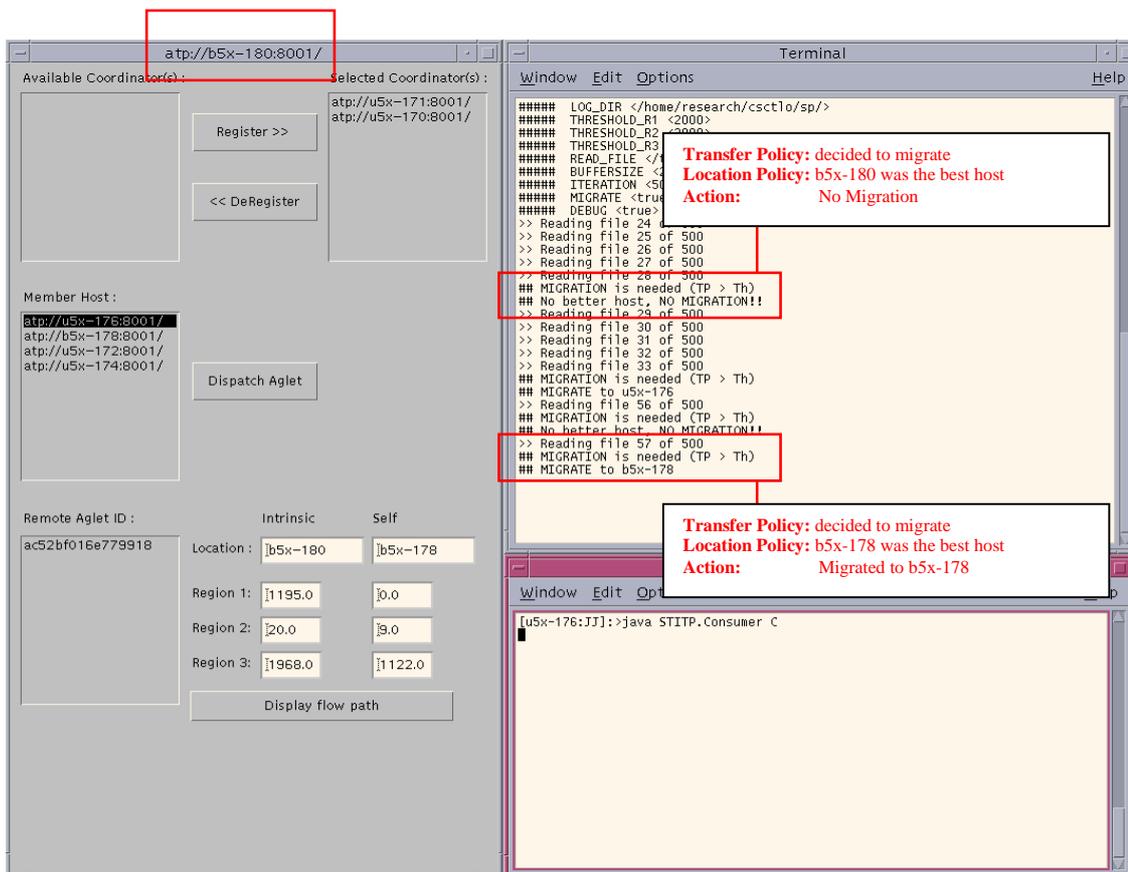


Figure 5.17. An example of when a migration decision was eventually materialized (from u5x-176 to u5x-178)

5.4.1 Leveraging Metrics and Their Statistics

The leveraging of metrics and their statistics by the SDITPM to support real-time migration decision is dynamic. The process requires no human interventions. The SDITPM is generic in the sense that it can be incorporated into any program module at will. For example, in the verification experiments it is incorporated into both the resident evaluator and the migrating logical server. Yet, it is important to ensure that the SDITPM indeed leverages the chosen metrics and their statistics correctly. The L and T windows (Figure 5.17) together help verify the following by cross referencing: a) when a migration decision was made, b) whether the decision was materialized, and c) if the last two issues were congruent with the sampled statistics and resolved correctly.

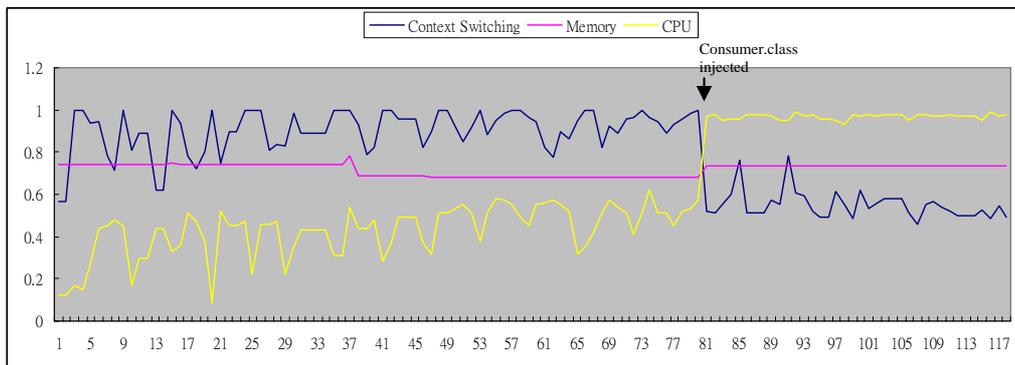


Figure 5.18. Consumer.class injects perturbations and sample remote statistics

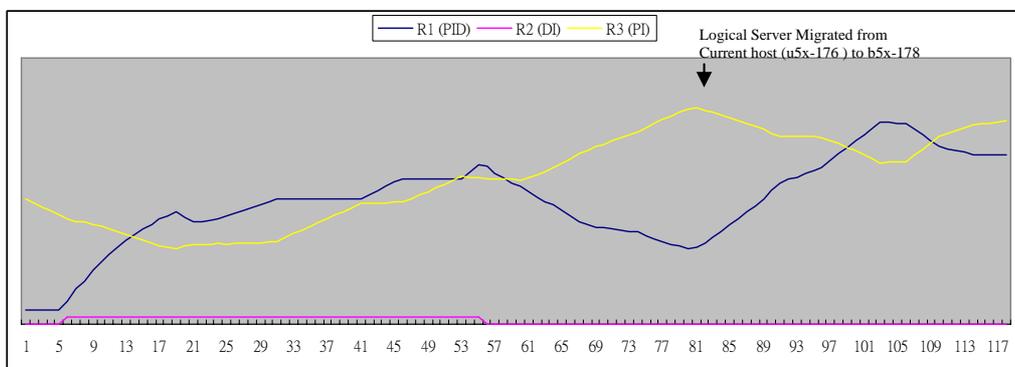


Figure 5.19. Intrinsic regional control values of captured in atp://u5x-176:8001/

In Figure 5.17, for example, several migration decisions were recorded by the L window but only one was actually materialized because eventually a suitable target node was found. Figure 5.18 shows the statistics for the three metrics, intrinsic context switching time, memory utilization and CPU utilization being leveraged by the resident evaluator in node `atp://u5x-176`. The point when the “Consumer.class” injected perturbations into the CPU utilization is marked. Figure 5.19 shows the immediate rise in the intrinsic R3 control value, which caused the migration of the logical server from node `u5x-176` to the target node `b5x-178`. The corresponding self control R3 value of the logical server that triggered the migration is also shown in Figure 5.20.

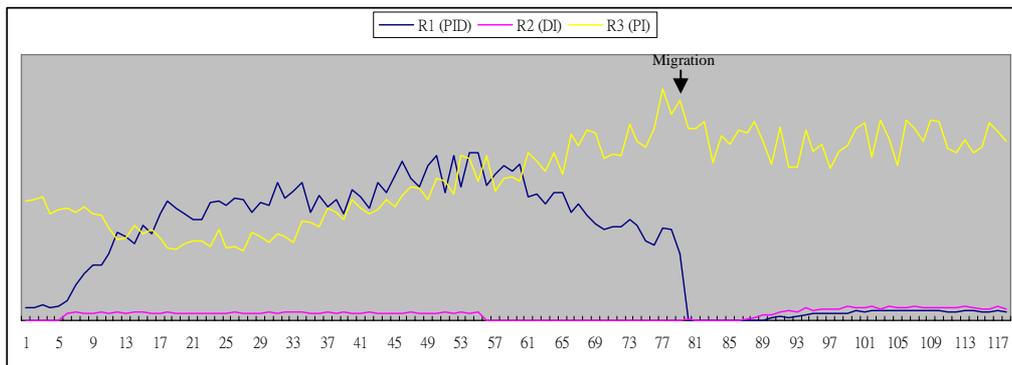


Figure 5.20. “Self” regional control values for the migration logged (from `u5x-175` to `b5x-178`)

Figure 5.21 compares the self TP_o value of the migrating logical server in Figure 5.17 with the intrinsic ones computed by the resident evaluators for the different villagers in the PI in the same period. At time T1, which corresponds to the actual migration from node `atp://u5x-176` to `atp://b5x-178`, node `b5x-178` had the lowest intrinsic TP_o value. Even though the migrating logical server leveraged the

same named metrics, CPU utilization, memory utilization, and context switching time or CST (in this case the logical server leveraged the self CST), the migrating server consistently produced a lower TP_o value most of the time as shown in Figure 5.22. This result is significant because it indicates that the logical server always travels to less busy nodes. That is, load balancing is indeed achieved by the object mobility steered by the SDITPM, which is a part of the migrating object construct.

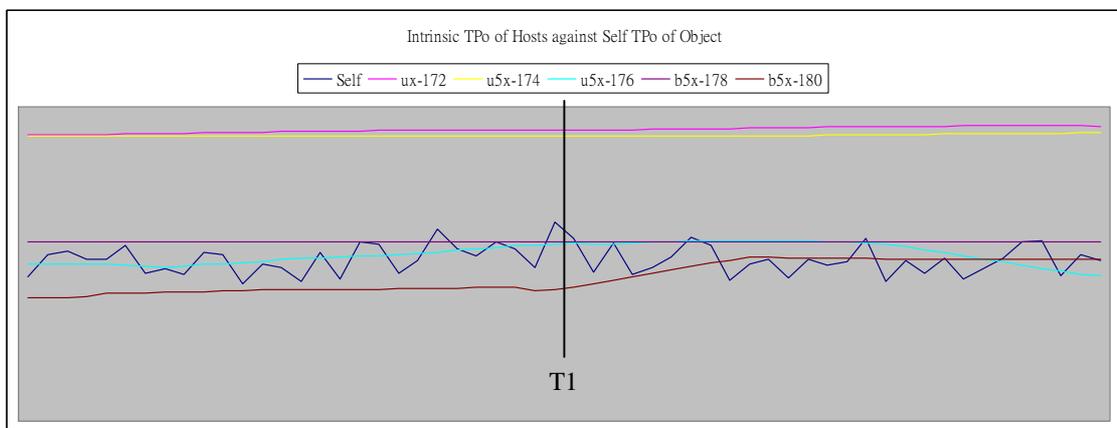


Figure. 5.21 Statistics at time T_1 when object migrated from u5x-176 to b5x-178 (refer to the L window in Figure 5.17)

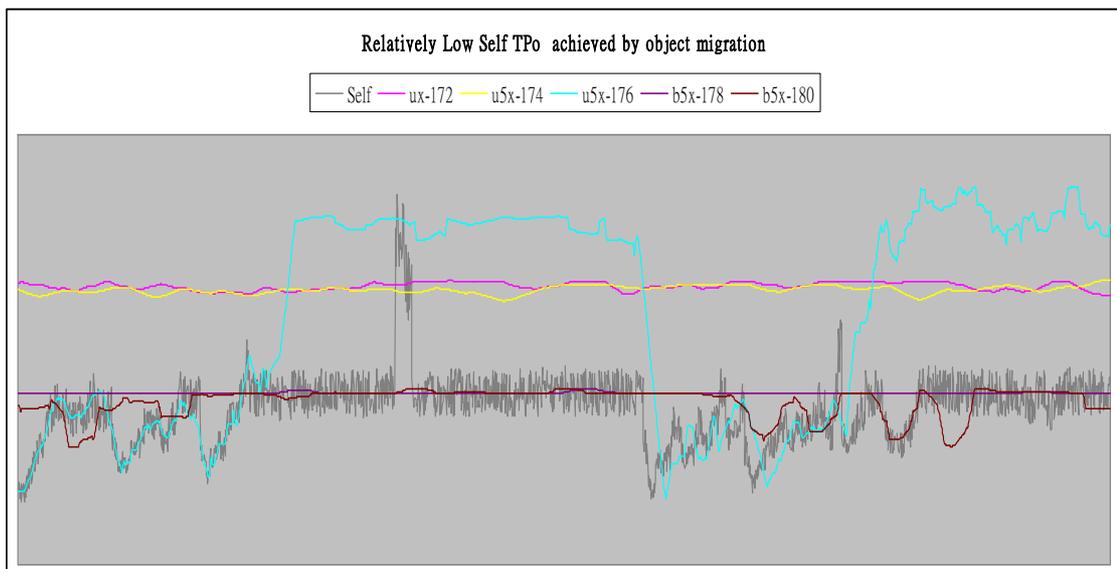


Figure 5.22 Comparing relevant TP_o values (refer to the PI in Figure 5.15)

5.5 Timing Analysis of the SDITPM Prototype

The novel SDITPM framework supports real-time transfer policy decision making by sampling and leveraging chosen metrics on the fly. It is important therefore for the SDITPM mechanism to have short execution time to avoid deleterious effects. If the SDITPM execution time is longer the duration of the problem, then the computed solution ends up correcting a historic or ghost problem, causing undesirable or deleterious effects.

SDITPMServer - VTune(TM) Performance Analyzer - [Static Code Analysis View - Java057C.JIT.mdb]

Executable: C:\Personal\MPH\research\final\SDITPM\VTune\Java057C.JIT

Symbols: C:\Personal\MPH\research\final\SDITPM\VTune\Java057C.JIT

Function Name	Offset	Length	Instructions	%Pairing	Penalties	Clocks	Warnings	Prefixed Instructic
CARate.<init>(V)	C1:00000000	80	27	67	5	38	-	-
CARate.addElement(D)V	1A8:00000000	71	26	31	7	43	-	-
CARate.getResult(D)	1A9:00000000	24	10	60	4	30	-	-
CATool.<init>(V)	C0:00000000	24	9	89	3	27	-	-
CATool.addElement(D)V	1A5:00000000	87	29	55	4	45	-	-
CATool.CA(D)	1A6:00000000	248	83	58	20	202	-	-
CATool.getResult(D)	1A7:00000000	24	10	60	4	30	-	-
CATool.init(I)V	C2:00000000	111	38	58	4	49	-	-
Init.<init>(V)	3A:00000000	2415	792	79	37	922	-	-
Init.getConfig(Ljava/lang/String;Ljava/lang/String;	59:00000000	48	18	67	-	21	-	-
Init.is(Ljava/lang/String;Ljava/lang/String;J	56:00000000	56	22	64	2	33	-	-
MetricDetector.<init>(Ljava/lang/String;V	8C:00000000	80	32	62	5	44	-	-
MetricDetector.getRef(Ljava/lang/String;D)	1A8:00000000	263	77	55	7	113	6	-
MetricDetector.getSample(I)D	1A4:00000000	127	48	46	11	75	-	-
MetricDetector.getTolerance(Ljava/lang/String;D)	1AC:00000000	263	77	65	7	108	2	-
SDITPM.<init>([[LMetricDetector;Ljava/lang/String;[[Ljava/lang/String;V	BD:00000000	856	203	71	18	261	-	-
SDITPM.convert(D)	1AA:00000000	176	59	44	18	146	4	-
SDITPM.getHostNames(Ljava/lang/String;	C3:00000000	144	58	66	10	76	-	-
SDITPM.getSample(D)I	1A3:00000000	56	18	78	-	20	-	-
SDITPM.init(D)V	8E:00000000	439	169	78	21	207	-	-
SDITPM.run()V	1A2:00000000	2096	609	60	96	919	2	-
SDITPM.sum()D	1AD:00000000	64	22	36	4	33	-	-
SDITPMServer.main(Ljava/lang/String;V	07:00000000	271	96	65	12	133	-	-
VmOnline.<init>(Ljava/lang/String;V	91:00000000	151	37	59	2	41	-	-
VmOnline.getHostNames(Ljava/lang/String;	98:00000000	144	58	66	10	76	-	-
VmOnline.run()V	8A:00000000	1424	475	55	89	892	11	-

Figure 5.23. One example of timing analysis of the SDITPM execution

The timing analysis by the Intel's VTune Performance Analyzer [VTune] shows that the Java SDITPM prototype needs an average execution time of 2050 clock/T cycles. Therefore, the speed of the SDITPM execution (S_{SDIT}) depends on

the speed in MHz of the hardware platform; that is, $S_{SDIT} = 2050 / \text{MHz}$. For a platform operating at 100MHz S_{SDIT} is $2050 / (100 * 10^6) = 20.5 * 10^{-6}$ seconds or 20.5 micro seconds. This is the SDITPM real-time limit for platform because SDITPM does not work properly for any problem duration shorter than 20.5 micro seconds. Figure 5.23 is a screen capture of the one of the VTune timing analyses for the SDITPM execution. The total number of clock cycles for the execution is the sum of the “Clocks” column within the inset boundary (i.e. the square – $1795 = 261 + 146 + 76 + 20 + 207 + 919 + 33 + 133$). It is relatively shorter than the average measured.

5.6 Connective Summary

Dynamic verification of the SDITPM needs the support of a cluster annexed from the Internet. This support is necessary to ensure that the dynamic SDITPM verification results are scalable to the open Internet. To achieve this, the previous Public Intranet concept is adopted with new distinctive features proposed. Timing analysis of the SDITPM execution was also carried out to find the limit of its real-time applicability because if the limit is exceeded deleterious effect may ensue. To simulate the presence of a location policy that finds the target for object migration, the principle of using the minimum TP_o value as the selection guide is adopted. The preliminary results from the dynamic verification exercise conclude that the novel SDITPM framework is indeed effective for load balancing through object mobility on the fly.

CHAPTER 6. CONCLUSION, FUTURE WORK AND ACHIEVEMENTS

In this paper the novel, original transfer policy framework, namely, the *statistical distribution independent transfer policy model* (SDITPM) for making sound migration decisions in a real-time manner, is proposed and verified. The four objectives stated at the outset are achieved, namely:

- a) Identify some useful performance metrics to support sound migration decisions made by logical servers in a real-time manner.
- b) Propose a technique (s) to efficaciously leverage metrics on the fly.
- c) Propose a scalable, non-intrusive conceptual framework that works at the transfer policy level to help logical servers make sound migration decisions; non-intrusive means no physical modifications of hosts are required.
- d) Verify the proposed conceptual framework.

The SDITPM mechanism should be incorporated as part of a logical entity (e.g. a logical server) construct and it steers the latter effectively in the migration process over a sizeable network such as the Internet. This mechanism leverages chosen system metrics on the fly, independent of their distribution types (e.g. long-range dependence (LRD) and short-range dependence (SRD)). That is, the SDITPM works by direct data measurement statistically, supported by the Convergence Algorithm (CA), which is derived from the Central Limit Theorem. Every system metric being leveraged is called a primary metric, from which the secondary metrics are derived. The present SDITPM framework is basically a PID or “P+I+D” controller (P for

proportional control, I for integral control and D for derivative control). The P and D control elements are secondary metrics and the I control sums their effects for the specific r^{th} control region.

From a primary metric, for example, the service request queue length Q the corresponding secondary metrics are derived. Every primary metric actually represents a plane/matrix of four control regions. Every region r^{th} is defined by the combination of the P and D control values at the time. If Q is considered, then the P control is the change in Q between two time points (e.g. t_1 and t_2) and the D control is the corresponding rate of change, $\frac{dQ}{dt} = \frac{(Q_{t_2} - Q_{t_1})}{(t_2 - t_1)}$. More precisely the control plane/matrix of a primary metric is defined by two *objective functions*, $\{0_p, \Delta_p\}^2$ as rows for P control and $\{0_d, \Delta_d\}^2$ as columns for D control. The four control regions defined by these objective functions are: a) Region 1 (i.e. r^1 or r^{PID}) defined by the $[P+, D+]$ value pair for PID control, b) Region 2 (i.e. r^2 or r^{DI}) defined by $[P-, D+]$ for “D+I” or DI control, c) Region 3 (i.e. r^3 or r^{PI}) defined by $[P+, D-]$ for PI control, and Region 4 (i.e. r^4 or r^{Inert}) defined by $[P-, D-]$ to be an inert or “don’t care” state. Now the regional control only takes the positive value(s) for P and D into account. When the effects of the D and P control are added for the regional control the summation is the integral control. In the SDITPM context the summation is planar integration for a single metric. If n metrics are leveraged simultaneously, then the current control effect for a region r_n^x is $C_{r_n^x} = \sum_{i=1}^n (P_i^x, D_i^x)$, where x indicates the control region (e.g. r^1 or r^{PID}). The (P_i^x, D_i^x) represents the P

and D deviation errors for the following objective functions respectively: $\{0_P, \Delta_P\}^2$ and $\{0_D, \Delta_D\}^2$. Conceptually, the deviation error ψ is the difference between the sample value S and the safety margin Δ (e.g. Δ for P control is Δ_P); $\psi = |S - \Delta|$ implies $\psi_P = |S_P - \Delta_P|$ and $\psi_D = |S_D - \Delta_D|$. To emphasize, the present SDITPM framework leverages only the following conditions: a) positive S and b) $S > \Delta$ for its P, D and I control elements. The migration decision depends on the dominant region, which should have the highest $C_{r_n^x}$ value at the time. Which control region is dominant is determined by the system dynamics. $C_{r_n^x}$ is called the regional transfer probability TP_r (i.e. $TP_r = C_{r_n^x}$), and the threshold for the region is Th_r . The dominant TP_r becomes the overall transfer probability TP_o and the Th_r value of the same region is now the overall Th_o threshold. The migration decision at the transfer policy level is affirmed for condition $TP_o > Th_o$. Whether the decision is materialized or not depends on the availability of a suitable target node for the logical server to migrate to. Finding the target node is the decision of the location policy, which is not within the scope of the present research. In the SDITPM verification exercise, however, the effect of the location policy is produced by using the A^4 visualization package, which is also part of the original contribution by this thesis

The experimental results with different primary metrics indicate that the Java SDITPM prototype can indeed effect transfer policy decisions that guide object migrations properly. To help visualize object mobility in migration processes in the experiments the original Object Migration Visualization Tool (OMVT), which is

also called the A^4 framework is proposed. This tool is based on the Public Intranet (PI) concept that allows objects within to roam freely. It also automatically records the object migration path over time. The next step planned for the SDITPM research is to identify more useful primary metrics and study how they can contribute to enabling the SDITPM mechanism to make sound transfer policy decisions over a more open Internet environment.

The major achievements by the thesis are as follows:

- a) A solid SDITPM framework that provides a good basis for deeper research in the area of network monitoring, management, and supporting real-time object migrations for the sake of load balancing over a sizeable network such as the internet.
- b) A SDITPM prototype that helps identify different useful metrics for making real-time object migration decisions at the transfer policy level. It requires very short execution time (only 2050 clock cycles) and this makes it immensely suitable for real-time applications.
- c) The experimental results confirm that the SDITPM mechanism can be easily incorporated as part of the construct of a logical server.
- d) Four publications so far help share the SDITPM experience with the international research community. The SDITPM findings contribute to enriching the body of knowledge of using object mobility to achieve load balancing and shorten the service RTT. These publications are as follows:

1. Jason C.T. Lo, Allan K.Y. Wong and Wilfred W.K. Lin, SDITPM: A Novel Transfer Policy Model to Facilitate Object Mobility that Shortens Service Roundtrip Time by Load Balancing over the Internet, to appear in the International Journal of Computer Systems Science & Engineering
2. Jason C.T. Lo, Allan K.Y. Wong and Wilfred W.K. Lin, A Novel Transfer Policy Model to Enhance the Serviceability of an E-Business, Proc. of the 2nd International Conference on E-Business and Telecommunication Networks (ICETE'05), Reading, United Kingdom, October 2005
3. Jason C.T. Lo, Allan K.Y. Wong and Wilfred W.K. Lin, SDITPM: A Novel Transfer Policy Model for Agent Server Migration in E-Business, Proc. of the 4th International Conference on Mobile Business, Sydney (ICMB'05), Australia, August 2005
4. May T. W. Ip, Jason C. T. Lo, Allan K. Y. Wong: Non-intrusive Protocol and Micro Convergence Algorithm Together Support Object Migration Over the Internet for Effective Load Balancing, Proc. of the conference of Parallel and Distributed Processing Techniques and Applications (PDPTA'02), Las Vegas, USA, 2002

To recap, overall the thesis has two original contributions as follows:

1. The novel SDITPM model: This PID control model, which can leverage any number of primary metrics, is original because of the following original conceptual elements:

- a. Formation of a control plane per primary metric defined by proportional (P) and derivative (D) controls.
- b. Planar and vertical/incidental integration for integral control.
- c. PID control that supports sound real-time object migration decisions at the transfer level, it is the first of its kind.

These original elements are built on the objective function concept, $\{0, \Delta\}^2$, which was proposed and successfully utilized before [GAC02]. The PID control concept has existed for a long time in industrial process control, but using it in the SDITPM to support effective real-time object mobility is a brand new concept.

2. The A^4 visualization tool: It traces the object mobility due to the dynamics of the SDITPM, which is part of the logical entity (server/agent/object) that migrates. This original tool was designed to support the SDITPM verification experiments. The A^4 design is based on the stable A^3 tool, which is Aglets based and was previously proposed for supporting aglet mobility. The A^3 tool, however, cannot support the SDITPM verification requirements, namely, explicit display of intrinsic and self regional transfer probabilities. Therefore, the A^3 architecture had to be changed to accommodate the newly proposed features to support the A^4 (i.e. Adapted A^3) functionality.

LIST OF ABBREVIATION

$0, 0_{sm}, 0_{sm}$	Steady-state reference (for primary/secondary metric)
A^3	Access security, Agent persistence and Avoidance of long queuing time for service
A^4	Adapted A^3
ANT	Average number of trials
BS	Barrier synchronization
CA	Convergence Algorithm
CF	Coordinator file
CI	Coordinator Interface
CORBA	Common Object Request Broker Architecture
CP	Coordinator.class program
CST	Context switching time
DCE	Distributed computing environment
DOC	Distributed object computing
DPE	Distributed process environment
DSA	Distributed service architecture
D^x	Derivative control value in region x
$I_{\psi_{l,d}}^r$ or $I_{\psi_{l,d}}$	Planar integration (for control region r)
IAT	Inter-arrival time
LRD	Long-range dependence
M^2RT	Mean Message Response Time

M^3RT	Micro Mean Message Response Time
MCA	Micro Convergence Algorithm
M_i	Sample mean of in cycle i
MPA	Migration probability analyzer
MUI	Member User Interface
NCT	Network configuration table
OMVT	Object migration visualization tool
PI	Public Intranet
PID	Proportional, integral and derivative
PVM	Parallel virtual machine
P^x	Proportional control value in region x
RPP	Remote Programming Paradigm
RT-NMM	Real-time network monitoring and management
RTT	Roundtrip time
r^x	Region x
$r(1,3)$, r^{PID} , $r^{C_3^1}$, or C_3^1	Region 1 of control plane (PID control region)
$r(2,3)$, r^{DI} , $r^{C_3^2}$, or C_3^2	Region 2 of control plane (DI control region)
$r(1,4)$, r^{PI} , $r^{C_4^1}$, or C_4^1	Region 3 of control plane (PI control region)
$r(2,4)$, r^{Inert} , $r^{C_4^2}$, or C_4^2	Region 4 of control plane (Inert state region)
SAP	Service access point
SDITPM	Statistical Distribution Independent Transfer Policy Model
SIMD	Single instruction multiple data
SLR	Simple linear regression

SMR	Split/migrate and redirect
SNMP	Simple Network Management Protocol
SRD	Short-range dependence
STK	Stack (number of primary metrics)
Th	Transfer threshold
Th_r	Regional transfer threshold
TINA	Telecommunication Information Networking Architecture
TPH	Transfer policy handler
TP_o	Overall transfer probability
TP_r	Regional transfer probability
TS	Tuple space
VUI	Visual User Interface
WWW	World Wide Web
Δ	Tolerance margin
λ	Inter-arrival rate at a queue
μ	Service rate by a server
ξ	Probability error that causes retransmission
ρ	Channel error probability
Ψ	Deviation error

BIBLIOGRAPHY

- [Accetta86] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian and M. Young, Mach: A New Kernel Foundation for UNIX Development. Proceedings of the Summer USENIX Conference, 1986, 93-112
- [AGLETS] IBM Aglets, http://www.trl.ibm.com/aglets/index_e.htm
- [Barak95] A. Barak, O. Laden and Braverman, The NOW MOSIX and its Preemptive Process Migration Scheme, Bulletin of the IEEE Technical Committee on Operating Systems and Application Environments, 1995, 7, 2, 5-11
- [Baratto00] A.M. Barotto, A. de Souza, and C.B. Westphall, Distributed Network Management Using SNMP, Java, WWW and CORBA, Journal of Network and Systems Management, 8(4), 2000, 483-497
- [Bivens99] A. Bivens, P. Fry, L. Gao, M. Hulber, B. Szymanski and Q. Zhang, Distributed Object-Oriented Repositories for Network Management, Proc. of the 13th Conference on Software Engineering, Las Vegas, USA 1999, 169-174
- [Bradshaw97] J.M. Bradshaw Ed., Software Agents, MIT Press, 1997
- [Braden98] B. Braden et al., Recommendation on Queue Management and Congestion Avoidance in the Internet, RFC2309, April 1998
- [Chatranon04] G. Chatranon, M. A. Labrador and S. Banerjee, A Survey of TCP-Friendly Router-Based AQM Schemes, Computer Communications, vol. 27, 2004, 1424-1440
- [Cheriton88] D.R. Cheriton, The V Distributed System. Communications of the ACM 1988, 31, 3, 314-333

- [Chin91] R.S. Chin and S.T. Chanson, Distributed Object-Based Programming Systems, ACM Computing Surveys, 23(1), 1991, 11-19
- [Cockayne97] W.T. Cockayne and M. Zyda, Mobile Agents, Manning Publication, Greenwich, Connecticut, 1997
- [Coulouris01] G. Coulouris et al, Distributed Systems - Concepts and Design, 3rd Edition, Pearson, 2001
- [Corradi00] A. Corradi, Parallel Objects Migration: A Fine Grained Approach to Load Distribution, Journal of Parallel and Distributed Computing, vol. 60, 2000
- [Cottrel99] L. Cottrel, M. Zekauskas, H. Uijterwaal and T. McGregor, Comparison of Some Internet Active End-to-End Performance Measurement Projects, <http://www.slac.stanford.edu/comp/net/wan-mon/iepm-cf.html> <<http://www.ulanr-net/viz/End2end>>, 1999
- [Crovella97] M.E. Crovella and A. Bestvros, Self-similarity in the World Wide Web Traffic: Evidence and Possible Causes, IEEE/ACM transactions on Networking, 5(6), Dec. 1997, 835-846
- [Dejan00] D.S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler and Songnian Zhou, Process Migration, ACM Computer Surveys, 32(3), September 2000, 241-299
- [Dillon00] Tharam S. Dillon, Allan K.Y. Wong and Wilfred W.K. Lin, A Model for Mobile-Agent-Based Distributed Computing over the Internet with Security and Dynamic Load Balancing, Proc. of the 3rd Pacific Rim International Workshop on Multi-Agents (PRIMA2000), Australia, 2000
- [Fuggetta98] A. Fuggetta, G.P. Picco and G. Vigna, Understanding Code Mobility, IEEE Trans. On Software Engineering, 24(5), May 1998, 342-361
- [GAC02] Allan K.Y. Wong, Wilfred W.K. Lin, May T.W. Ip and Tharam S. Dillon, Genetic Algorithm and PID Control Together for Dynamic Anticipative

- Marginal Buffer Management: An Effective Approach to Enhance Dependability and Performance for Distributed Mobile Object-Based Real-time Computing over the Internet, *Journal of Parallel and Distributed Computing (JPDC)*, vol. 62, Sept. 2002, 1433-1453
- [Gartner99] F.G. Gartner, *Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments*, *ACM Comp. Survey*, 31(1), 1999, 1-26
- [Gelernter92] D. Gelernter and N. Carriero, *Coordination Languages and Their Significance*, *Communications of the ACM*, 35(2), 97-107
- [Goscinski91] A. Goscinski, *Distributed Operating Systems - The Logical Design*, Addison-Wesley, Reading, MA, 1991
- [Gosling96] J. Gosling, B. Joyo and G. Steele, *The Java Language Specification*, Addison Wesley, 1996
- [Herrate91] V. Herrate and E. Lusk, *Studying Parallel Program Behavior with Upshot*, Technical Report, ANL-91/15, Argonne National Laboratory, 1991
- [Holt00] A. Holt, Long-range dependence and self-similarity in World Wide Web proxy cache references, *Proc. of the IEE Communications*, 147(6), Dec. 2000, 317-321
- [Iepm] L. Cottrel et al., *Comparison of Some Internet Active End-to-End Performance Measurement Projects*,
<http://www.slac.stanford.edu/comp/net/wan-mon/iepm-cf.html>
- [Ip01] May T.W. Ip, Wilfred W.K. Lin, Allan K.Y. Wong, Tharam S. Dillon and Dian Hui Wang, *An Adaptive Buffer Management Algorithm for Enhancing Dependability and Performance in Mobile-Object-Based Real-time Computing*, *Proc. of the IEEE ISORC*, 2001, Magdenburg, Germany, May 2001, 138-144

- [Ip03] T.W. Ip, Development of a Micro IEPM (Internet End-to-End Performance Measurement) Techniques for Internet Based Computing, MPhil Thesis, Department of Computing, Hong Kong PolyU, 2003
- [Jain91] R. Jain, The Art of Computer Systems Performance Analysis, Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley 1991
- [Joo01] Y. Joo, V. Riberio, A. Feldman, A. Gilbert and Willinger, TCP/IP Traffic Dynamics and Network Performance,: A Lesson in Workload Modeling, Flow Control, and Trace-Drive Simulations, SIGCOMM Computer Communications Review, 31(2), April 2001
- [Karray02] F. Karray, W. Gueaieb and S. Al-Sharhan, The Hierarchical Expert Tuning of PID Controllers Using Tools of Soft Computing, IEEE Trans. on Systems, Man, and Cybernetics, Part B, 32(1) February 2002, 77-90
- [Kihl97] M. Kihl, C. Nyberg, H. Warne and P. Wollinger, Performance Simulation of a TINA Network, Proc. of the Globecom'97, Phoenix, USA, 1997, 1567-1571
- [Lakshman97] T. Lakshman and U. Madlow, The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss, IEEE/ACM Transactions on Networking, 5(3), 1997
- [Lewis96] T. Lewis, The Next 10,000 Years: Part I, IEEE Computer, April 1996, 26-32
- [Lewandowski98] S.M. Lewandowski, Frameworks for Component-based Client/Server Computing, ACM Computing Surveys 30(1), March 1998, 3-27
- [Lin05a] Wilfred W. K. Lin, Allan K. Y. Wong, Richard S.L. Wu , and Tharam S. Dillon, A Novel Real-Time Self-Similar Traffic Detector/Filter to Improve the Reliability of a TCP Based End-to-End Client/Server Interaction Path

- for Shorter Roundtrip Time, ICETE'05, Reading UK, October 2005, vol. 1, 94 - 101
- [Lin05b] Wilfred W. K. Lin, Allan K. Y. Wong, and Tharam S. Dillon, Application of Soft Computing Techniques to Adaptive User Buffer Overflow Control on the Internet, IEEE Trans. of Systems, Man and Cybernetics, Part C, 2005
- [Litzkow88] M. Litzknow, M. Livny, M. Mutka and M. Condor, A Hunter of Idle Workstations, Proc. of the 8th International Conference on Distributed Computing Systems, 1988, 104-111
- [Lo02] May T. W. Ip, Jason C. T. Lo and Allan K. Y. Wong: Non-intrusive Protocol and Micro Convergence Algorithm Together Support Object Migration Over the Internet for Effective Load Balancing. PDPTA 2002: 836-842
- [Lo05a] Jason C.T. Lo, Allan K.Y. Wong and Wilfred W.K. Lin, A Novel Transfer Policy Model to Enhance the Serviceability of an E-Business, to appear in the 2nd International Conference on E-Business and Telecommunication Networks, Reading, United Kingdom, pp.160-164
- [Lo05b] Jason C.T. Lo, Allan K.Y. Wong and Wilfred W.K. Lin, SDITPM: A Novel Transfer Policy Model for Agent Server Migration in E-Business, Proceedings of the 4th International Conference on Mobile Business, Sydney, Australia, pp.581-584
- [M2RT] Allan K.Y. Wong, Tharam S. Dillon, Wilfred W.K. Lin and T.W. Ip, M2RT: A Tool Developed for Predicting the Mean Message Response Time for Internet Channels, Journal Computer Networks, vol. 36, 2001, 557-577
- [M3RT] Allan K.Y. Wong, May T. W. Ip, Tharam S. Dillon: M3RT: An Internet End-to-End Performance Measurement Approach for Real-time Applications with Mobile Agents. ISPAN 2002: 119-124

- [Medina00] A. Medina, I. Matta and J. Byers, On the Origin of Power Laws in Internet Topologies, ACM SIGCOMM, 30(2), 2000
- [Mitrani87] I. Mitrani, Modelling of computer and communication systems, Cambridge University Press, 1987
- [Mitrani98] I. Mitrani, Probabilistic Modelling, Cambridge University Press, 1998
- [Molnar99] S. Molnar, T.D. Dang and A. Vidacs, Heavy-Tailedness, Long-Range Dependence and Self-Similarity in Data Traffic, Proceedings of 7th International Conference on Telecommunication Systems, Modelling and Analysis, March 1999, Nashville, USA, 18-21
- [Myers90] B.A. Myers, Programming, programming by Example, and Program Visualization: A Taxonomy, in Visual Programming Environment - Paradigm and Systems, E.P. Gilnert Eds., IEEE Press, 1990, 33-40
- [Oda01] K. Oda, S. Tazuneki and T. Yoshida, The Flying Object for an Open Distributed Environment, Proc. of the IEEE 15th International Conference on Information Networking (ICOIN-15), Beppu City, Japan, January 2001
- [OMG00] Object Management Group, The Common Object Request Broker: Architecture and Specification, Version 2.4, October 2000
- [OMG96] Common Object Request Broker Architecture and Specification, Object Management Group, Document Number: 96.03.04, 1996
- [OMG97] Object Management Group, White Paper on CORBA as an Enabling Factor for Migration from IN to TINA: A P508 Perspective, OGM DTC Document: Telecom/97-01-01, January 1997
- [Ousterhout88] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson and B. Welch, The Sprite Network Operating System, IEEE Computer, 1988, 23-26
- [Ousterhout94] J. Ousterhout, Tcl and the Tk Toolkit, Addison-Wesley Longman, 1994

- [P4] R. Butler and E. Lusk, User's Guide to the P4 Parallel Programming System, Technical Report, ANL-92/17, Argonne National Laboratory, USA, October, 1992
- [Padhye98] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, Modeling TCP Throughput: A Simple Model and its Empirical Validation, Proc. of the ACM SIGCOMM '98 Conference, Vancouver, Canada, 1998
- [Paxson95] V. Paxson, S. Floyd, Wide area traffic: the failure of Poisson Modelling, IEEE/ACM Transactions on Networking, 3(3), June 1995, 226 -244
- [Paxson99] V. Paxson, End-to-end Internet Packet Dynamics, IEEE/ACM Transactions on Networking, 7(3), June 1999, 277 -292
- [PID01] May T.W. Ip, Wilfred W.K. Lin, Allan K.Y. Wong, Tharam S. Dillon and Dian Hui Wang, An Adaptive Buffer Management Algorithm for Enhancing Dependability and Performance in Mobile-Object-Based Real-time Computing, Proc. of the IEEE ISORC'2001, Magdenburg, Germany, May 2001, 138-144
- [PVM90] V.S. Sunderam, PVM: A Framework for Parallel Distributed Computing, Concurrency: Practice and Experience, vol. 2, 1990, 11-19
- [PVM93] A. Benguelin, J. Dongarra, W. Jiang, R. Manhek and V.S. Sanderam, PVM3 User's Guide and Reference Manual, Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, USA, May 1993
- [Rashid81] Rashid, R. and Robertson, G. Accent: a Communication Oriented Network Operating System Kernel. Proceedings of the 8th Symposium on Operating System Principles, 1981, 64-75
- [RDM00] Allan K.Y. Wong and Richard S.L. Wu, 5E: A Framework to Yield High Performance in Real-time Data Mining over the Internet, Proc. HPCAsia'2000 Conference, Beijing, P.R. China, May 2000

- [RFC793] RFC 793 - Transmission Control Protocol, IETF, September 1981
- [Rosen92] W. Rosenberry, D. Kenney and G. Fisher, Understanding DCE. O'Reilly & Associates, Inc, 1992
- [Shirazi95] B.A. Shirazi, A.R. Hurson, K.M. Kavi, Scheduling and Load Balancing in Parallel and Distributed Systems, IEEE Computer Society Press, 1995, 309-313
- [Stankovic98] J.A. Stankovic et al., Deadline Scheduling for Real-Time Systems - EDF and Related Algorithms, Kluwer, 1998
- [Tanenbaum03] A. S. Tanenbaum, Computer Networks, 4th Edition, Prentice Hall, 2003, 550-553
- [Taqqu03] M.S. Taqqu, Fractional Brownian Motion and Long-Range Dependence, in Theory and Applications of Long-Range Dependence, P. Doukhan et al., Eds., Birkhuser 2003, 5-38
- [Thottan98] M. Thottan and C. Ji, Proactive Anomaly Detection Using Distributed Intelligent Agents, IEEE Network, 12(5), Sept-Oct 1998, 21-27
- [VTune] Intel VTune Performance Analyzer,
www.intel.com/support/performance/vtune/v5
- [White97] J. White, Telescript Technology: An Introduction to the Language, White Pager, General Magic, Inc., Sunnyvale, CA, 1997 (also appeared in J. Bradshaw, Software Agents, AAAI/MIT Press)
- [Wong00] Allan K.Y. Wong, Wilfred W.K. Lin and Tharam S. Dillon, Local Compilation: A Novel Paradigm for Multilanguage-Based and Reliable Distributed Computing over the Internet, Special Issue: Mobile & Wireless Communications & Information Processing, Journal of Simulation, 75(1), July 2000, 18-31

- [Wong01] Allan K.Y. Wong and Joseph H.C. Wong, A Convergence Algorithm for Enhancing the Performance of Distributed Applications Running on Sizeable Networks, The International Journal of Computer Systems, Science & Engineering, vol. 16, no. 4, July 2001, 229-236
- [Wong01A3] Allan K.Y. Wong, Wilfred W. K. Lin: The A 3 Fault-Tolerant Framework for Distributed Computing with Multiple Mobile Agents over Sizeable Networks. ICOIN 2001: 355-360
- [Wong01Words] Allan K. Y. Wong, Tharam S. Dillon, May T. W. Ip, Wilfred W. K. Lin: A Generic Visualization Framework to Help Debug Mobile-Object-Based Distributed Programs Running on Large Networks. WORDS 2001: 240-250
- [Wong96] A.K.Y. Wong and D.S. Yeung, RHS - A Framework for Exploiting Distributed Parallelism Efficiently, The International Journal of Computer Systems, Science & Engineering, 11(3), 1996, 177-184
- [Wong99] Allan K.Y. Wong and Tharam S. Dillon, A Fault-Tolerant Data Communication Setup to Improve Reliability and Performance for Internet-Based Distributed Applications, the Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing (PRDC'99), December 1999, Hong Kong (SAR), P.R. China, 268-275
- [Wu99] Kun-Lung Wu and Philip S. Yu, Load Balancing and Hot Spot Relief for Hash Routing among a Collection of Proxy Caches, IBM Research Report RC21420, March 1999
- [Xu00] C.Z. Xu and B. Wims, A Mobile Agent Based Push Methodology for Global Parallel Computing, Concurrency: Practice and Experience, v.12, 2000

- [Yeung98a] D.S. Yeung and A.K.Y. Wong, An Evaluation Identifying Features of Unified Distributed Programming, The International Journal of Computer Systems, Science & Engineering, 13(5), September 1998, 311-322
- [Yeung98b] D.S. Yeung and A.K.Y. Wong, The OORHS: A Conceptual Framework that Provides Easy and Reversible Distributed Programming, The International Journal of Computer Systems, Science and Engineering, 13(5), September 1998, 289-302
- [Zajcew93] R. Zajcew, P. Roy, D. Black, C. Peak, P. Guedes, B. Kemp, J. Lo, M. Leibensperger, M. Barnett, F. Rabii and D. Netterwala, An OSF/1 UNIX for Massively Parallel Multicomputers, Proc. of the Winter USENIX Conference, 1993, 449-468
- [Zhou94] S. Zhou, X. Zheng, J. Wang, and P. Delisle, Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems, Software Practice and Experience, 1994