The Use of Agent to Incorporate Network Awareness into Dynamic Proxy Framework: An Overview

KhongNeng Choong, Borhanuddin M. Ali, Veeraraghavan Prakash, Elok R. Tee*
Department of Computer and Communication, Faculty of Engineering
*Department of Communication Technology and Networking, Faculty of Computer Science
Universiti Putra Malaysia
43400 Serdang, Selangor, Malaysia
and
YokeChek Yee
Global Software Group, Motorola Multimedia Sdn Bhd
16th Floor Menara Luxor, 6B Jalan Persiaran Tropicana
47410 Petaling Jaya, Selangor, Malaysia

ABSTRACT

It has been observed that research in mobile agent focused on the development of platforms and on the application of the concept. Network awareness is one of the applications of mobile agent, concerning of how agent determine and make the most efficient use of network resources. This paper describes the role and functionality of mobile agent in the context of a dynamic proxy framework named the Chek Proxy Framework (CPF). Attention is paid on setting out a multi-agent-based framework to enrich CPF with a Network Awareness Module (NAM), which is implemented in an agent named ObjectBasket (OB). NAM is a single framework that integrates resource discovery, load monitoring and migration, and fault management, to operate in a network with dynamic proxy servers. The main objective of NAM is to deliver robustness and best-effort QoS guarantees into the existing CPF system adaptively, based on the availability of resources in the network. In this paper, we present the architecture of NAM, status generation algorithm, implementation rules, and the management of faults and overloading.

Keywords: dynamic proxy, agent, network awareness, application level active network

1. INTRODUCTION

CPF is an application level approach that deals with the growing Internet, by deploying intermediate object called the Dynamic Application Proxy Server (DAPS) into the network. DAPSs are placed dynamically at runtime on nominated voluntary client hosts by the central server, to execute tasks on behalf of the central server. DAPS scales the central server and mitigate the bandwidth consumption by reducing the number of direct client-to-server connections over the WAN.

The uniqueness of CPF lies on the use of client machines to host DAPS services. This is done by appointing selected clients (as proxy) that have already downloaded, or are downloading contents, to turn around and serve the contents to other local clients at runtime, thereby relieving the server and network from redundant loads while accelerating the speed of delivery. Contents can be of both static and dynamic nature such as web pages and video streams. Tapping into the large availability of client resources in a secure manner offers several advantages, including the reduction of server loads (with distributed DAPS), masking of DAPS failures (given large number of client machines) and possibly exploiting the use of local IP multicasting.

Recent works on accelerating content delivery include the End-to-End Media Paths [1], TPS [2] and Conductor [3]. In such systems, all intermediate nodes work in concert to deliver contents. Their main studies include resource discovery, service coordination and service interference for optimizing end-to-end resource consumptions. Less emphasis is made on the robustness, failure recovery, load balancing and migration aspects of the systems. In [1], attention is paid only on route and code selection, leaving issues of robustness, usability and programmability behind. The work on TPS focuses on confining proxy service regions and solving service coupling and interference matters. Unlike CPF, there is no notion of agent in these works on enriching the programmability, flexibility and functionality of the framework to accommodate various application requirements.

The rationale of CPF is to migrate workloads from the central server to the DAPS running in client networks. Extending to such idea, the challenges tackled in this paper are to provide load migrations and fault managements within the client networks (among the DAPS and other voluntary clients), given that the clients' availability is changing over time, i.e. they arrive and leave the session at arbitrary time. Attention is also paid on when load migrations should be initiated since frequent migrations could cause service interruptions to client applications due to the DAPS handover and client redirection processes. It is believed that the proper setup is to minimize load migration unless the current DAPS is overloaded or failed. The next challenge is to derive a flexible and scalable model that delivers the above features with proper interfaces to the existing CPF architecture.

Throughout this paper, we highlight the architecture and protocol of NAM in fulfilling these requirements. For reasons of adaptability and programmability, we incorporate NAM into OB, as a way to bridge NAM to CPF by leaving the underlying CPF architecture untouched. Our focus is on designing NAM as a single framework for performing resource discovery, and both load and fault management. We set out to equip OB with the ability of delivering and setting up application objects, and fortifying CPF with load migration and fault tolerance capabilities. However, we do not yet have sufficient experience with NAM implementation to comment on its usability, programmability, cost, and performance.

The paper is organized as follows. In Section 2, we look at various reasons for deploying agent in CPF, particularly on the functionality and benefits of agent

on meeting the requirements of the CPF distributed model such as object delivery and creation, interface design and security. Section 3 presents the Network Awareness Model (NAM), describing how network awareness capability is embedded in our agent, i.e. the OB. Section 4 concludes the paper with a glimpse on our future works.

2. THE USE OF AGENT IN CPF

The idea of OB is motivated by [4], where a small boot-strap applet that includes the necessary logic to direct the installation process and manage the local resources such as disk access permission, for caching and executing the carried objects. With the incorporation of network awareness capability, OB reveals the characteristics of agent, i.e. by being able to sense its environmental changes and act on them to meet some objectives without human intervention. However, we do not address our OB as mobile agent because the degree of mobility (range limit) is restricted to only one single hop due to its unjustified mobile functionality and partly due to the security reason [5].

Instead of adopting simple object invocation techniques as in Application Level Active Network (ALAN) [6] and Java Applet, CPF utilizes agent as the application setup coordinator for several reasons. First, because the grouping of clients to a particular proxy in CPF follows the dynamic network clustering policy defined by the CPFserver, the delivery and creation of application objects have to be managed and undertaken judiciously by an agent with reactive behavior. Second, the invocation of DAPS depends on various factors, including the client machine load, client voluntary preference, client network locations and client size. Hence, instant and periodic decisions have to be made on activating and demising DAPS appropriately. The third reason to adopt agent is its ability to sense the networks autonomously for potential proxy voluntaries and take proactive operations to foster load balancing, load migration, and fault management in the event of DAPS failure. This includes monitoring the network path from the clients to the access Internet Service Provider (ISP) network, to discover prospective voluntaries.

As an installation coordinator, agent is responsible for managing the version and upgrading of object to ensure application objects can always communicate with one another without conflicts. The use of agent is regarded efficient because the execution environment of CPF (CPFnode) needs only to verify the digital signature of the agent to infer the validity of the application objects it carries. Based on the principle of trusts, this approach shortens the application setup process as the signed agent is only verified once regardless of the number of objects it holds. In our design, OB is delivered only once from the central server to the CPFnode and this incurs only one-time cost for an exchange of performance improvement for the CPF-enabled applications.

Although alternatives exist such as embedding the features discussed above as plug-in modules in the CPFnode, we believed that agent offers a better software engineering model, as it helps to conceptualize solutions better, improve code modularity and reusability. Agent paradigm also keeps designers shielded from the intricacies of network, system and protocol heterogeneity. The CPFnode is thus designed to manage only the resource access and communication routines with the underlying OS. On the security aspect, choosing agent allows us to reference existing security measurements as in [5].

3. INCORPORATION OF NETWORK AWARENESS INTO CPF

Network sensing capability is crucial to CPF for acknowledging any relevant changes in the availability of network resources (i.e. the coming-in and dropping-off of voluntary clients), and reacting accordingly based on the CPF clustering policy to guarantee performance of any CPF-enabled applications. CPF-enabled applications are to be driven by the framework to deliver their services in an efficient and dynamic manner. With effective network awareness, we plan to further study in the future, the deployment issues of adopting CPF to run on both wired and wireless networks.

In general, network-aware systems are designed according to three main criteria. Firstly is a comparative study on both active and passive monitoring approaches over the amount of control traffic generated. Next is an investigation on the pros and cons of both centralized and distributed status information collection techniques. Lastly is on the monitoring frequency, which generally could be performed in either an on-demand or continuous basis.

In this section, we describe the NAM based on the above three criteria. The major parts of the discussion are:

- **Brief overview of CPF:** Describe the general framework of CPF.
- **Architecture of NAM:** Explain the functionality and interactions of NAM components.
- **NAM hand-shaking protocol:** Provide step-by-step explanations on how NAM initiates and operates.
- **Heartbeat message:** Describe the format, algorithm, and cost of adopting the heartbeat messaging system in NAM.
- **Implementation rules:** Highlight the rules that govern when and how NAM should be applied.
- **Management of fault and overloading:** Discuss how NAM fortifies CPF in event of proxy failure and proxy overloading.

Brief overview of CPF

Figure 1 gives an overview of how agent is adopted in CPF. It shows the client machine hosting the DAPS as the local server for all local subnetwork clients, including the DAPS-hosting client itself. The subnetwork is defined as an enterprise network that sits behind a local gateway router. DAPS conserves backbone bandwidth by utilizing only a single in- and out-bound connection to/from the subnetwork.

There are basically two types of OBs that are different in terms of object size and responsibility. The first type is called the Heavy OB (HOB) because it carries both the DAPS and client application objects. HOB is configured to the requirement of delivering and creating application of both DAPS and client tiers. HOB sits on the machine that hosts the DAPS services. The size of HOB is larger than the Light OB (LOB), the second agent type. LOB holds no application objects and it is delivered when there is an

active proxy service (the presence of HOB) in the local subnetwork of the client. The LOB works economically by contacting the local HOB for a copy of the client application object.

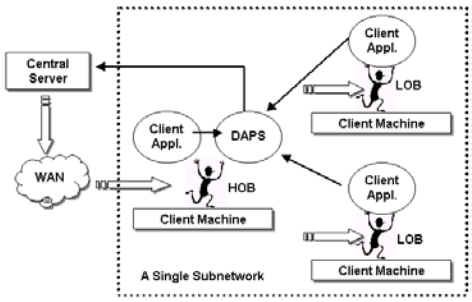

Figure 1: Agent-based dynamic proxy framework

Architecture of NAM

Our first attempt is to incorporate modules into the CPF for discovering potential hosts in the networks (the path from the LAN to the ISP accessed network) that allows agents to move, clone and coordinate as a means to conduct load migration. The subsequent attempt is to strengthen CPF with fault tolerance capability, where the agent is able to immediately allocate new DAPS session upon detecting the failure of the present one. To fulfill the above two requirements, CPF has to be equipped with intelligence that perceives resource variations in the network and react autonomously and proactively. Figure 2 depicts an architectural overview of the NAM, which consists of three basic modules namely the Metric Management Module (MMM), Decision Making Module (DMM) and Application Management Module (AMM). This model is incorporated into the existing OBs (both HOB and LOB) to instill network-awareness capability.

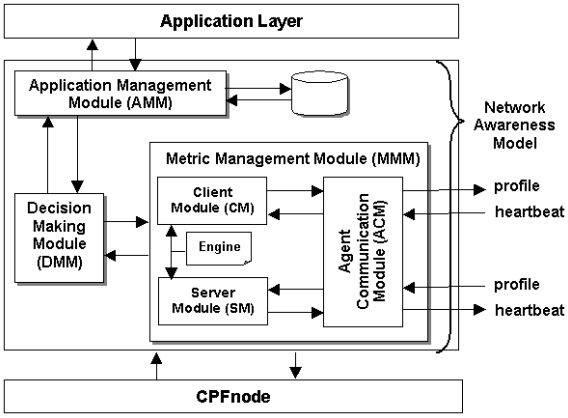

Figure 2: The Network-Awareness Model (NAM)

In general, the OB that is responsible for collecting network status information acts as the NAM-Server (NS), and the OB that is being monitored is known as the NAM-Client (NC). NS serves two roles: network monitor and directory service. As a monitor, NS guards the active DAPS from overloading and failure by ensuring that the DAPS is always on, free from excessive loads and has backup DAPS candidates. This is only possible by monitoring the status of the DAPS host and potential DAPS candidates. NS is also given the ability to stop a particular DAPS session due to performance dissatisfaction provided more capable DAPS candidates participate into the session. With a handful of status information, NS is ready to provide a directory service, listing the type and location of DAPS services.

NC, the corresponding node to NS, relies on NS to obtain services and perform request redirections. In return, NC must periodically update the NS of its status because each host where the NC runs has a chance to be selected as backup DAPS candidate. Together, both NS and NC form a multi-agent scenario that works in a cooperative manner.

The MMM is further made up of three sub-modules, i.e. the Server Module (SM), Client Module (CM) and Agent Communication Module (ACM). Both the SM and CM are the corresponding engines that drive the services offered by both NS and NC.

The role of the SM is to sense the presence of other OBs in the network. SM adopts an active monitoring approach to collect information on node failure, node properties and node distance. Passive monitoring is not used here because it measures only the Internet regions where application traffic traversed, and thus unable to discover the distance to some unknown but potential networks/hosts. Other limitations of passive monitoring can be found in [7]

For reasons of flexibility, SM is designed to work along with various metric collection engines, so as to widen the choice of different collection methods. The choice of selecting a method depends on the network location where NS is deployed. As example, NS at network edge (i.e. the ISP), which is known as the parent NS could learn the distance from other OBs over the Internet in a much simpler and efficient manner, with the help of third party Internet "weather" services such as IDMaps [7].

Besides collecting metric information from external services, NS is also responsible for notifying its corresponding NCs about its own status, status of the DAPS-hosting machine, and information of DAPS candidates as backups. Such information are encapsulated into a message called heartbeat. NS must relay its heartbeat to its child NS across the ISP network to enterprise network, or multicast to its local NCs if both were within the same local network segment as shown in Figure 3. Communications between the headquarters and remote office adopt the same methodology.

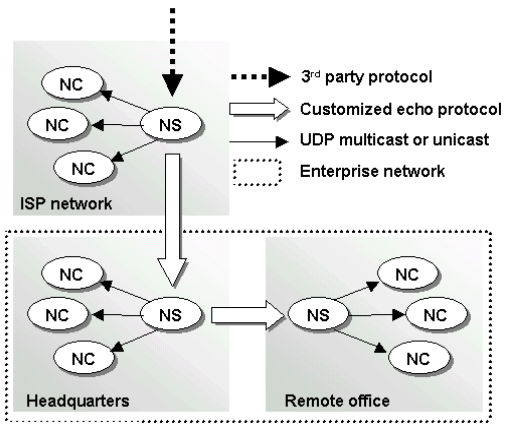

Figure 3: Heartbeat distribution in NAM

We adopt a distributed approach in collecting and storing status information to avoid single point of failure as in the centralized approach. Explanation of how NAM helps to tackle faults is discussed later. In summary, NS at each network level is responsible for sending heartbeat messages to local NCs, collecting status message of local NCs, and for ensuring the parent NS is always active. For reason of efficiency, NS is further required to shortlist NCs (those capable of hosting) and releases those that are not from submitting their profiles (which include the machine capacity, workload, network latency etc.) to the NS in future. However, all NCs must always consult the latest heartbeat from the NS.

CM performs the opposite functionality to SM as depicted by the In and Out arrows in Figure 2. Its main job is to update the NS with its current profile and collect from NS the periodic heartbeats. CM consults the specific metric collection engine as in SM in order to work synchronously with the SM at the other network end.

The role of the ACM is to establish network connections to other corresponding NS, or NCs, or to other network service portals based on the metric engine adopted by either the CM or SM. It is responsible for sending and receiving messages (either profile or heartbeat) in a format that is compatible to both the sender and the corresponding receiver. Once a profile is received by the SM, it is logged and passed to the DMM for further processing. ACM utilizes the underlying system channel that was established by the CPFnode to perform active monitoring.

DMM is designed to infer and make decisions based on the collected messages and rules defined by the CPF. These rules are adhered to determine the subsequent actions that the MMM should take. As an example, if it were discovered that the current DAPS is overloaded or failed, DMM of this host would instruct the MMM to activate the backup (DAPS candidate). This backup is chosen among all voluntary clients and has the highest CPU capacity with the lowest utilization within the local network segment.

The third module, AMM, performs its job based on the decision made by the DMM. As an example, upon detecting the failure of the current HOB, DMM would inform AMM to launch a backup DAPS to continue the proxy services (assuming the current host (itself) holds the highest DAPS candidacy). For audio application where packet missing is intolerable, multiple DAPSs could be allocated at different hosts to deliver simultaneous streams to safeguard against packet loss. It is also important for AMM to relay the performance status of the DAPS to the DMM as a way to ensure DAPS is never overloaded.

NAM hand-shaking protocol

NC and NS interact with each other along the time line diagram as depicted in Figure 4. Each voluntary client must go through such interactions. The interactions start according to the following procedures:

1. Each NC is initially registered with its local NS by submitting a static profile. This static profile contains several properties of NC such as its URL, machine capacity of its host machine, and past DAPS hosting records (if any).
2. Upon receiving a NC registration request, NS replies by sending a brief heartbeat message with only the information of the current DAPS and NS.
3. Subsequently, NC would response by sending a dynamic profile that consists of its current workload, network latency (from the NS) etc. to the NS.
4. Upon receiving the dynamic profile, NS first logs the information to the disk and further determines which few nodes should be periodically sensed in the future. As part of this process, a list of nodes is identified and accumulated as more NCs participate in the session. These selected nodes are those that will be appointed as backup DAPS if the current DAPS failed or overloaded.
5. In the subsequent distribution of heartbeat messages, URLs of those selected nodes are appended.
6. NC is required to always consult the latest heartbeat message from the NS. Selected NCs (backups) as stated in the heartbeat message is required to update the NS periodically with their dynamic profiles. For NCs that were not short-listed, they are fed with heartbeats and are excluded from updating the NS.

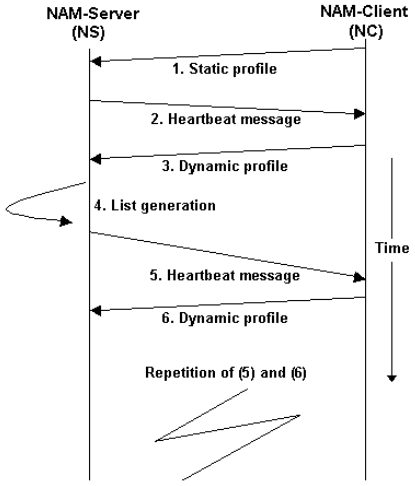

Figure 4: The hand-shaking protocol of NAM

Heartbeat message: format, generation and cost

Heartbeat format: The heartbeat message consists of a list of URLs that is divided into two parts: the URL of DAPS and URL of NS. Let U = { all CPF clients in a particular network segment }, $A \subset U \wedge B \subset U$, A = { a | a is a list of backup candidates for DAPS } and B = { b | b is a list of backup candidates for NS }, $\forall\, a\ (a \in A \wedge a \notin B)$, $\forall\, b\ (b \notin A \wedge b \in B)$. The entry size of each part is such that $|A| \geq |B|$, meaning that the DAPS availability (contents delivery) has more weight than the presence of NS (performance/faults monitoring). As shown in Figure 5, the first entry of each part in the heartbeat refers to the respective service that is currently active. This is followed by a series of $|A|$ -1 or $|B|$ -1 entries, which states the URL of backup services.

We suggest the threshold value of $|A|$ and $|B|$, denoted by $threshold_a$ and $threshold_b$ to be either 3 or 4. The value is kept small to avoid both processing and transmission overheads during heartbeat generations and transmissions. Keeping a list beyond 4 entries is

regarded infeasible for two reasons. Firstly, there is no guarantee that all backup entries would remain valid (available) over the session period, as voluntary clients could disconnect from the network at anytime. Secondly, not every client is eligible to host DAPS.

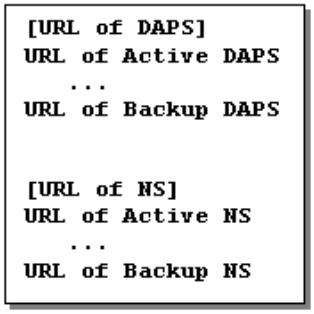

Figure 5: Format of the heartbeat message

Heartbeat generation: The list is compiled in ascending order according to one or more selected properties of the target nodes (found in both static and dynamic profiles). Properties include the mean of cumulative voluntary period (*VP*), machine capacity (*MC*), machine utilization (*MU*), and past service hosting period (*HP*). The difference between *VP* and *HP* is that, the formal refers to the amount of voluntary periods that the client has offered since the first execution of its copy of CPFnode software, whereas the latter refers to the total time spent on hosting DAPS. The *MC* refers to a threshold that is defined in finite scales (could be based on the CPU speed, memory size or a combination of both) to induce the range of serving capability of the voluntary machines. As an example, by scaling the machine capacity into 5 levels, level-0 denotes machine with the highest capacity while level-4 represents the opposite extent.

Figure 6 shows the algorithm that generates the heartbeat message based on these properties.

$\forall\, i$ (i | i is a voluntary client)
  if ($VP_i > Threshold_{vp}$) $\wedge$ ($MC_i > Threshold_{mc}$)
    then $Q \leftarrow Q \cup \{ i \}$

sort Q according to *MU;*
  then by the *HP*, both in ascending order

$R \leftarrow$ first x number of clients in Q;
  where $x \leftarrow min(|\sum \forall i| , Threshold_a + Threshold_b)$

$l \leftarrow m \leftarrow n \leftarrow 0$
$S[l] \leftarrow$ URL of current DAPS
$S[ceil((|R|+2)/2)] \leftarrow$ URL of current NS

while ($n < |R|$)
  $l \leftarrow ((++m) * ((n+1)\%2==1)) +$
    $((ceil((|R|+2)/2) + m) * ((n+1)\%2==0))$
  $S[l] \leftarrow R[n]$
  $n \leftarrow n + 1$

Figure 6: Heartbeat generation algorithm

The algorithm consists of two parts. The first part generates an intermediate list that contains the first x number of backup nodes. This list excludes the entries of the current DAPS and NS. The second part extracts each entry in the list to form two separate lists of both NS and DAPS.

In the first part, we use two equally important factors to serve as the requirements for backup candidates, i.e. the mean of cumulative voluntary period ($Threshold_{vp}$) and machine capacity ($Threshold_{mc}$). The voluntary period allows us to gauge the availability and thus reliability of the node, whereas the machine load helps to guarantee performance. Nodes are collected as long as they satisfy the thresholds given by both $Threshold_{vp}$ and $Threshold_{mc}$. The collected nodes are then sorted according to machine utilization. This is to ensure that lightly loaded nodes are put at higher priorities. Given that a few nodes might be of equivalent workloads, performing a secondary sorting based on the mean hosting period helps to place less loaded nodes with fewer hosting periods at higher priority.

The reasons for putting nodes with fewer hosting periods in front of the list is for costing purposes. Since CPF relies on the concept of voluntary computing, the main motivation to attract more volunteers is by means of *hosting rewards*, i.e. volunteers are paid for hosting services. In such reward scheme, the hosting period determines the amount of rewards. Longer hosting periods incur higher cost to the content providers, but yield more incentives to the volunteers. Hence, keeping nodes of equivalent loads but of lower hosting periods at higher priority gains, results in acceptable performance with minimal hosting costs.

As described earlier, only a few nodes should be inserted into the heartbeat to reduce processing and transmission overheads. This is realized by x, which takes the minimum value of either the total client size, or the summation of both $threshold_a$ and $threshold_b$. These thresholds state the maximum number of the backups for each DAPS and NS section. The sorted and minimized list is then constructed, denoted by R.

The last part is a loop that traverses through R to form two virtual sub-lists represented by the list S at the size $|S| = |R| + 2$. The upper portion of S that is of size $p = ceil(|S|/2)$ is used by DAPS backups while the remaining lower portion is occupied by NS backups at the size of $q = floor(|S|/2)$ such that $p \geq q$. The current DAPS and NS will not be replaced if they are not overloaded, and thus their entries will stay at the first position in each portion.

If either of them were overloaded or failed, the most capable backup candidates at each portion will be appointed to take over the responsibilities. Details of the handover shall be discussed in our subsequent paper. In cases where R is empty, i.e. no qualified candidates, NS would distribute the heartbeat that consists of the same entries as before, i.e. only with the current NS and DAPS.

Execution cost: The algorithm is executed at fixed time interval based on the most recent dynamic profiles collected from all NCs. NS collects these profiles based on the continuous monitoring technique, instead of the on-demand technique to increase the sensitivity of NS to clients' availability and status changes. Doing this keeps service interruptions to the CPF-enabled applications at a minimal level in the event of DAPS failure, at the expense of frequent monitoring traffics.

There are 2 types of monitoring costs, namely the horizontal (*hc*) and vertical costs (*vc*). These costs are measured in unit of message injected into the network. The *hc* incurs whenever nodes that stay within the same network segment communicates, i.e. sending or receiving either the heartbeat (*h*) or profiles (*p*). In short, *hc* is a function of both *h* and *p* (i.e. the volume of 2 types of messages), as shown in the following equation:

$$hc = |U|*h + |I|*p \quad (1)$$

where *U* is a set of all CPF clients in a particular network segment, and *I* is composed of both *R* and *R'*, i.e. the first *x* number of voluntary clients that meet the proxy hosting requirements (See Figure 6), and those do not, respectively. Such cost is usually insignificant given the speed of today's Ethernet connectivity.

The *vc* refers to overheads produced by communicating nodes situated across different segments, e.g. LAN to ISP, or LAN to LAN over the Virtual Private Network (VPN). It grows slightly slower than the linear rate, i.e. $log|I| < |I|/C_i < |I|$, where C_i refers to the capability of the specific NS machine *i*, in serving requests. This growing rate is also applicable to the DAPS allocations. In brief, NAM allows only 2 vertical connections to be made, i.e. one for contents delivery (DAPS) and another for monitoring (NS) purpose.

In summary, potential backup nodes include those that have high CPU power, light workloads, and possess hosting records, i.e. have hosted DAPS in a reliable manner before.

Implementation rules

As explained earlier, the allocation of NS and NC is triggered by the activation of the internal modules in the MMM, i.e. the SM and CM, respectively. There are cases where both CM and SM are active at the same time. In short, there are three models that guide how both interact as shown in Figure 7.

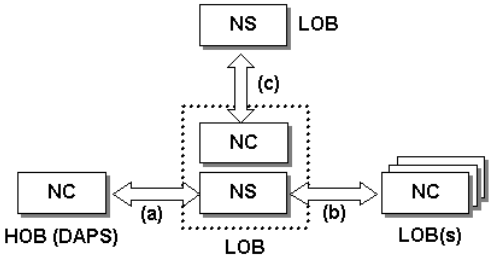

Figure 7: The NAM interaction models

In general, the interaction between NS and NC starts as soon as there are two voluntary clients in the network. Based on the current proxy allocation policy in CPF, the first voluntary client will be selected as DAPS and regarded as the NC. Serving as NC allows the performance of DAPS to be constantly monitored by the NS, which is served by the second client (LOB). Such NS and NC relationship adopts the first model depicted in Figure 7, arrow (a).

The reason of separating the execution of DAPS and NS into different hosts is to avoid overloading the DAPS host with workloads incurred by NS. The second reason is to avoid a complete failure if the single host that serves both services fails. We apply the segregation of tasks to ease proxy handover process by isolating the status management routines of NS into different hosts.

In cases where DAPS is found overloaded, the second client could take over the task of DAPS only if the second client is more capable than the DAPS host. Doing this will incur a twist of both NS and NC identity between the respective clients. If the DAPS fails, the second client either form a direct connection to the central content server, or connect to a parent NS for further redirections (if the parent NS is available).

The second model as depicted in Figure 7, arrow (b) applies to clients (NCs) that interact with their local NS in a LAN environment. This model applies when the LAN has at least three clients, i.e. the DAPS (HOB), NS and NC (LOB). If either the DAPS or NS fails, there is now an extra NC that could take over the responsibility.

The last model as shown in Figure 7, arrow (c) shows a cascading relationship between the NS and NC, with both situated at different network levels. Here, the LOB at lower hierarchical level serves two identities, i.e. NS (for its local clients) and NC (to its parent NS). In the event of parent NS failure, NC could refer to other parent NS backup(s) listed in the most recent heartbeat message. In scenario where the DAPS in the ISP network fails or overloads, parent NS could activate the backup DAPS (if there is any). Otherwise, the parent NS would just identify other parent DAPS (through other NSs) to serve all subordinate client requests.

In a network where none of the clients are volunteers, multiple connections shall be made directly to the central content server, and no instance of DAPS, NS or NC will be instantiated. However, if DAPS services are discovered in the network neighbourhood (ISP or other subnetworks) within the performance and security boundary, client requests will be redirected accordingly.

In cases where there is only one CPF-enabled client, the respective NS is activated to connect to external NS, which could be in the parent network or the Internet. Such connection style is categorized under the third model.

Management of fault and overloading

Fault management: NAM is designed to protect CPF from two types of failures, i.e. DAPS and NS failures, which essentially occur in the client network. We do not emphasized fault issues of the central server because existing distributed models such as the server farms and replications (passive and active) are readily adoptable as solutions.

Given either the DAPS or NS has failed, they are basically 3 ways to perform recovery. The first or coarse-grained solution is to get all clients to reconnect to the central server. Second is to instruct only selected clients to reconnect to the central server, and shares their subsequent downloaded contents with other clients. The third approach is to redirect the selected clients to reconnect to their immediate predecessors (e.g. DAPS or NS in the ISP network segment), rather than to the central server. NAM uses

the third approach, i.e. by getting the clients to reconnect to some "stand-by" backup node, dictated in their most recently received heartbeat. This allows the clients to resume their interrupted sessions without incurring a sudden surge of workload to the central server. To guarantee a seamless and smooth service recovery, both backup DAPS and NS should be previously allocated but configured to a passive state, waiting to be invoked at later time. Such approach is similar to the traditional techniques in distributed systems such as the hot backups, and object group replication with virtual synchrony.

In network where there are only 2 clients, i.e. the DAPS and NS. The termination of either client would cause the remaining client to link directly to its immediate predecessor.

Overloading management: As noted, the workload of NS increases as more voluntary clients join the CPF session. Besides distributing heartbeats, and receiving heartbeats from parent NS, the local NS has to make decision on filtering and selecting capable voluntary clients. If there were any client that is more capable than the present DAPS, in terms of CPU power and memory capacity, the NS would call for a proxy handover session. The selected voluntary client shall then be promoted to serve as the new DAPS, and the current DAPS will be discarded once every client has been redirected to the new DAPS. Future clients shall also be redirected to this new DAPS until another possible proxy handover is initiated.

Appropriate proxy handover helps to deliver certain degree of QoS guarantees to the CPF-enabled applications. This is made possible by monitoring the current DAPS while not ignoring the potentiality of other voluntary clients to host DAPS. However, to avoid performance degradation and unnecessary overheads, proxy handover is initiated only if the present DAPS is near to the overloading state, instead of depending merely on the availability of better resources. Proxy handover could be driven by a linear cost model, i.e. by averaging and accumulating the cost of serving each connection (by executing a small benchmark program on the CPFnode), followed by performing an *n*-step ahead prediction where total cost $(TC) = TC + n$, where $TC < T_{max}$. T_{max} is the maximum service capability of the respective DAPS. This allows us to conduct simple prediction on future workloads. However, such prediction is only possible if DAPS is hosted on a dedicated voluntary machine, unlike those in SETI [8], where user's computations (launching of applications, etc.) could interrupt any ongoing services in the background.

4. CONCLUSIONS AND FUTURE WORKS

We have proposed an agent called ObjectBasket (OB) to serve as an application setup coordinator with network awareness capability in a dynamic proxy framework named the CPF. A Network Awareness Module (NAM) that is aims to foster load monitoring, load migration and fault management is also presented. We have explained how NAM serves as a single model to fortify CPF with migration, failure and performance transparency, and integration to the application layers on delivering best-effort QoS guarantees.

The challenges to efficient and effective agent design are the tradeoffs between usability and cost. Including all the described features into the agent could make the agent bulky and less mobile, thus slowing down the delivery process. It is envisioned that future improvements would be on partitioning partial responsibilities of the agent to the CPFnode for reason of performance. The ultimate objective is to produce a lightweight agent that is both effective and efficient. With the incorporation of network-awareness functionality, we believe DAPS could be allocated efficiently and yet effectively to serve requests with minimal service interruptions.

In any distributed system like this, security will be an issue. A malicious host could tap into any ongoing CPF session and claim to be the valid NS in order to gain control of redirecting client requests. A subsequent and probably more serious problem is that, given a malicious NS, clients may be redirected to a malicious DAPS that delivers fake contents in response to client requests.

We do not yet have sufficient experience with NAM to comment on its usability, programmability, cost, and performance, however, we have started addressing its security flaws, and planned to evaluate the feasibility and performance of NAM with prototype implementations in the near future.

5. REFERENCES

[1] A. Nakao, L. Peterson and A. Bavier, "Constructing End-to-end Paths for Playing Media Objects", Elsevier Publ., Computer Networks, Vol.38, No.3, 2002, pp. 373-389.

[2] B. Knutsson and L. Peterson, "Transparent Proxy Signaling", Journal of Communications and Networks, Korean Institute of Communication Sciences (KICS) Vol.3, No.2, 2001.

[3] M. Yarvis *et al.*, "Conductor: A Framework for Distributed Adaptation". IEEE Workshop on Hot Topics in Operating Systems, 1999, pp. 44-49.

[4] T. Sundsted, Alternative Deployment Methods, JavaWorld.com, July 2000.

[5] M.S. Greenberg *et al.*, "Mobile Agents and Security", IEEE Communications Magazine, Vol.36, No.7, 1998.

[6] A. Ghost, M. Fry and G. Maclarty, "An Infrastructure for Application Level Active Networking", Elsevier Publ., Computer Networks, Vol.36, No.1, 2001, pp. 5-20.

[7] P. Francis *et al.*, "IDMaps: A Global Internet Host Distance Estimation Service", IEEE/ACM Transactions on Networking, Vol.10, No.4, 2001, pp. 525-540.

[8] SETI@home, available at http://setiathome.ssl. berkeley.edu, 2000.