A Collective Approach to Harness Idle Resources

Sachin Goyal and John Carter

UUCS-08-009

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

September 25, 2008

Abstract

We propose a collective approach for harnessing the idle resources (cpu, storage, and bandwidth) of nodes (e.g., home desktops) distributed across the Internet. Instead of a purely peer-to-peer (P2P) approach, we organize participating nodes to act collectively using *collective managers* (CMs). Participating nodes provide idle resources to CMs, which unify these resources to run meaningful distributed services for external clients. We do not assume altruistic users or employ a barter-based incentive model; instead, participating nodes provide resources to CMs for long durations and are compensated in proportion to their contribution.

In this paper we discuss the challenges faced by collective systems, present a design that addresses these challenges, and compare it with previous approaches. We show that the collective service model is a useful alternative to the pure P2P models. It provides more effective utilization of idle resources, has a more meaningful economic model, and is better suited for building legal and commercial distributed services.

A Collective Approach to Harness Idle Resources

Sachin Goyal and John Carter
School of Computing, University of Utah
{*sgoyal, retrac*}*@cs.utah.edu*

Abstract

We propose a collective approach for harnessing the idle resources (cpu, storage, and bandwidth) of nodes (e.g., home desktops) distributed across the Internet. Instead of a purely peer-to-peer (P2P) approach, we organize participating nodes to act collectively using *collective managers* (CMs). Participating nodes provide idle resources to CMs, which unify these resources to run meaningful distributed services for external clients. We do not assume altruistic users or employ a barter-based incentive model; instead, participating nodes provide resources to CMs for long durations and are compensated in proportion to their contribution.

In this paper we discuss the challenges faced by collective systems, present a design that addresses these challenges, and compare it with previous approaches. We show that the collective service model is a useful alternative to the pure P2P models. It provides more effective utilization of idle resources, has a more meaningful economic model, and is better suited for building legal and commercial distributed services.

1 Introduction

Modern computers are becoming progressively more powerful with ever-improving processing, storage, and networking capabilities. Typical desktop systems have more computing/communication resources than most users need and are underutilized most of the time. These underutilized resources provide an interesting platform for new distributed applications and services.

We envision a future where the idle compute, storage, and networking resources of cheap network-connected computers distributed around the world are harnessed to build meaningful distributed services. Many others have espoused a similar vision. A variety of popular peer-to-peer (P2P) services exploit the resources of their peers to implement specific functionality, e.g., Kaaza [17], BitTorrent [8], and Skype [14]. The large body of work on distributed hash tables (DHTs) exploit peer resources to support DHTs (e.g., Chord [15], Pastry [24], Tapestry [31], and CAN [22]), on top of which a variety of services have been built. SETI@home [25] and Entropia [7] farm out compute-intensive tasks from a central server to participating nodes.

We propose a new way to harness idle resources as managed "collectives". Rather than a purely P2P solution, we introduce the notion of *collective managers* (CMs) that manage the resources of large pools of untrusted, selfish, and unreliable *participating nodes* (PN). PNs contact CMs to make their resources available, in return for which they expect to receive compensation. After registering with a CM, each PN runs a virtual machine (VM) image provided by the CM. CMs remotely control these VMs and use their processing, storage, and network resources to build distributed services. Unlike projects like Xenoservers [23], *a CM does not provide raw access to end node's resources to external customers*. A CM is an application service provider, aggregating idle resources to provide services like content distribution and remote backup. Figure 1 illustrates a possible use of a collective to implement a commercial content distribution service that sells large content files (e.g., movies, music, or software updates) to thousands of clients in a cost-effective way.

The Collective uses an economic model based on currency. A collective manager earns money in return for providing services and then shares its profits with PNs in proportion to their contribution towards differ-

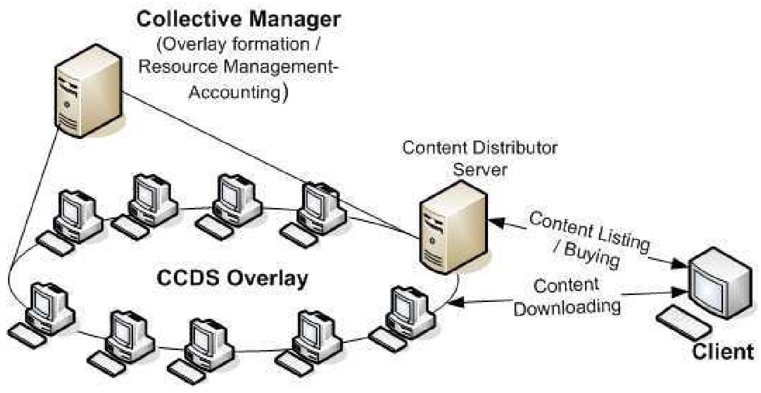

Figure 1: Collective Content Distribution Service

ent services. The basic unit of compensation is a CM-specific credit that acts as a kind of currency. Users can convert credits to cash or use them to buy services from the CM or associated partners.

Since collectives may include selfish nodes, it is important to mitigate the neagative effects of selfishness. Selfish nodes can resort to cheating for earning more than their fair share of compensation. Cheating behavior has been observed extensively in distributed systems, e.g., free riding in Gnutella [1] and software modifications to get more credit than earned in SETI@home [16]. To mitigate negative impact of selfishness, we propose to employ economic deterrents comprised of an offline data-analysis-based accounting system and a consistency-based incentive model. Services are designed specifically to facilitate these economic deterrents, and use security mechanisms to ensure accountability across different transactions. While we cannot prevent users from cheating, our mechanisms mitigate cheating behavior by making it economically unattractive.

The rest of the paper is organized as follows. In Section 2, we disuss the target environment and different approaches for addressing the problem. In Section 3, we compare collective approach to existing work. In Section 4 we do a simple cost analysis of a collective system and compare it with self managed clusters and utility computing systems (amazon s3/ec2 [2]). In Section 5, we describe the architecture of a collective system, its security infrastructure, and its incentive model. We then conclude in Section 6.

2 Design Space

Our target environment consists of potentially millions of end-nodes distributed all across the Internet. They are untrusted, and unreliable - i.e., they suffer from frequent node churning as well as failures. Each node can exhibit selfish behaviors. Our goal is to use unutilized resources of these end-nodes to build meaningful *commercial services*. Our design is similar to a firm in traditional economic systems, where multiple people come together to work for a common goal. Our incentive and interaction model is neither based on altruism, nor on the bartering.

A system based on altruism relies on *altruistic* users that provide services without any incentive, or provide more than the required service to others. For example, in the context of file sharing P2P systems, altruistic users share the files on their computer even though they do not get anything in return. Others remain in the system after downloading a file and upload the data to other users even when they are not required to do so. Systems like gnutella and Kazaa were based on altruism. These systems eventually suffer from freeloading, which degrades the quality of the network substantially. More importantly, these systems do not have an incentive model and thus are more useful for free or illegal content distribution instead of a commercial system.

Bartering is the exchange of goods or services between two people in a mutually beneficial way. One can use bartering based mechanisms to provide incentives for interaction. We categorize bartering systems into two categories - lazy bartering and active (currency-based) bartering. We use the term *"lazy bartering"* for systems where people participate in the system only long enough to perform a particular transaction such as downloading a song. As stated above, this leads to the underutilization of a node's resources unless the node's administrator is altruistic. BitTorrent is based on *lazy bartering*. To overcome this underutilization problem, one may try *active bartering*, that is using bartering to maximally utilize available resources (not just when a node needs some service in return). Using currencies as an intermediary in bartering provides the necessary tool for achieving that. But it is still tough to find eligible users with whom to barter. One is limited by the goods (e.g., music/movies) that one has for sharing and by the availability of other users interested in paying currency for the objects one owns. Additionally the inherent untrusted and unreliable nature of end nodes makes currency-based bartering risky For example, it is easy for a cracker to use a currency-based bartering mechanism to acquire resources at multiple nodes, and then use them for network attacks. Also, individual nodes can provide only a limited amount of resources, so anyone interested in building a service like a content distribution system will have to find and make bartering agreements with many end nodes. This process has to be repeated whenever a new service is built.

The collective model provides a useful approach to handle these problems in a simple but realistic manner. First, in a collective, a participating nodes shares its resources using a VM instance and provides root access to that instance to the CM. Only the trusted collective manager has direct access to the VM running on a participating node, so we do not need to worry about an unknown party using the nodes for nefarious purposes, e.g., to launch network attacks. Second, the collective system uses the long term association with nodes and the presence of a CM to counteract the untrusted and unreliable nature of PNs. A collective manager provides a simple service model to potential partners/customers, who do not have to worry about inherent chaos of a system built out of end nodes.

Another way to think about a collective is by comparing it to a person having $10000 of savings. He/she can either deposit their savings in a bank and get paid interest, or he/she can lend it to other people, potentially getting a higher rate of return than provided by banks. Like bartering in P2P, personal lending suffers from a trust problem; what if borrower does not return your money? Additionally one has to search for potential borrowers and the savings remain unutilized during the search period.

We can compare a collective to the formation of organizations in real life. As human civilization has progressed, there has been a clear move towards forming organizations - whether it is universities, banks, manufacturing plants, or other commercial organizations. While we still have freelancers, the majority of productive work is performed by well defined organizations.

In some ways, a collective resembles the collections of zombie machines often used by malicious users to launch distributed denial of service attacks. We differ from "zombie nets" in that we only use the resources of willing participants and allow PNs to limit how their resources are used (e.g., no more than X kilobytes-per-second of network bandwidth).

3 Related Work

Our collective model, while similar in certain aspects to previous work, differs in a number of important ways. There are four main domains of related projects that also deal with utilizing the resources of computers distributed across the Internet. The first is peer to peer systems like bittorrent [8], gnutella [11] etc. The second is compute-only services like seti@home [25], entropia [7] etc. The third is utility computing systems like Xenoservers [23]. The fourth is grid computing systems [9].

Peer to Peer Services:

Unlike typical P2P systems, we do not assume that PNs are altruistic (e.g., Kazaa [17] or Gnutella [11]) or willing to "barter" their resources in return for access to the end service (e.g., BitTorrent [8]). Rather, PNs make their resources available to CMs to build distributed services, in return for which they are compensated by CMs.

Using idle resources to run *arbitrary* services, rather than only services that the local user uses, improves resource utilization. A node's compute and network resources are perishable — if they are not used, their potential value is lost. In an incentive model that employs bartering, e.g., BitTorrent, nodes typically participate in the system only long enough to perform a particular transaction such as downloading a song. At other times, that node's idle resources are not utilized unless the node's administrator is altruistic. In the collective, a CM will have much larger and more diverse pools of work than personal needs of individual participants; thus a CM will be better able to consume the perishable resources of PNs. PNs, in turn, will accumulate credits for their work, which they can use in the future however they wish (e.g., for cash or for access to services provided by the CM). In a sense, we are moving the incentive model from a primitive barter model to a more modern currency model.

Additionally in a collective the CM has direct control over the VMs running in participating nodes. This control can be utilized to provide a predictable service to the customers, e.g., the CM can dynamically change the caching patterns in response to sudden demand.

Distributed Compute Intensive Services:

Unlike seti@home [25] and Entropia [7], the idle storage and networking resources of PNs can be harnessed, in addition to idle processing resources. As a result, collectives can be used to implement distributed services (e.g., content distribution or remote backup) in addition to compute-intensive services. These services have different design chanllenges than compute intensive services.

First, seti@home or other compute-only services are embarrassingly parallel and does not require any interaction between different nodes. Services like content distribution or backup services require cooperation from multiple nodes to successfully cache/replicate a piece of content, and to provide service to the customers. Handling these interactions (e.g., multiple node collusion) while still being able to manage selfish behaviors is a much different and tougher problem than handling embarrassingly parallel applications.

Second, applications like content distribution or remote backup require timely delivery of service to customers – thus adding a real time response component. There are no similar real-time requirements in seti@home-like applications.

Third, applications like content distribution or remote backup require different mechanisms and design to detect selfish behaviors by participating nodes.

Utility Computing Systems:

Utility computing systems like Xenoservers [23], Planetlab [20], and SHARP [10] deal with similar problems, but these systems handle resource sharing at the granularity of VMs, and are not bothered about the design and challenges of building a service using those resources.

For example, unlike collective they do not provide solutions for service level selfish behaviors by the participating nodes (or sites). Many of these assume trusted nodes. Projects like SHARP [10] assume that the service managers have some external means to verify that each site provides contracted resources and that they function correctly (assumption 7 in their paper [10]).

The focus of these projects are dedicated powerful servers of high reliability. While in collective, our main focus is to harness the idle resources of unreliable end-nodes.

Similar to these projects, PNs in a collective exploit VM technology for safely running arbitrary untrusted code provided by CMs. But unlike utility computing projects like Xenoservers or SHARP, we do not provide raw VMs to external clients. We allocate only one VM on a node, and run multiple services inside that one VM. In other systems any untrusted third party can acquire VMs and potentially use them for nefarious activities like network attacks. In contrast, our one VM per participating node is only controlled by the trusted collective manager.

Grid Computing Systems:

Systems like Condor [18] manage the idle resources of collections of nodes, but typically manage the resources of *trusted* nodes that remain in the "pool" for substantial periods of time. In contrast, we assume that PNs are unreliable (frequently fail or leave the collective) and are non-altruistic (will do as little productive work as possible while maximizing their compensation).

Systems like computational grids [9] also deal with distributed resources at multiple sites, though again their main focus is on trusted and dedicated servers.

4 Cost Analysis - Clusters, Utility Computing, and Collective

Modern computers have become quite powerful over the years, and typically have more processing, storage, and communication resources than most user need, and remain underutilized. A verification of this can be seen from the success of systems like seti@home, Gnutella, Kazaa, and BitTorrent. Based on data from Jan 2004 to June 2004, CacheLogic reported that peer-to-peer systems (P2P) consume 80% or more of the traffic for last mile service providers [6]. Another study from CacheLogic put the P2P percentage of Internet traffic at about 60% at the end of 2004 [5]. The same study reports that BitTorrent was responsible for as much as 30% of all Internet traffic at the end of 2004 [5].

Our collective system intends to harness these unutilized resources of already deployed computers. Our target nodes can be either home desktops or computers deployed in communities or enterprise environments. These nodes are bought and deployed to serve some specific purpose, but their resources are not utilized 100% all the time. The goal of the collective system is to harness these unutilized resources to build commercial services, and distribute the profits back to participating nodes.

Instead of using unutilized resources of end-nodes, one can potentially use a cluster of PCs to provide similar resources. In this subsection, we do a quick quantitative analysis to understand the opportunity cost of a collective in comparison to the cluster approach. We first estimate the resource capabilities of a collective system consisting of one million nodes. We then use those estimates to calculate the potential cost for building an equivalent cluster using a self-managed and a utility computing approach.

Assumptions: In this analysis, we assume that 10% resources of a given node are available when the end-user is actively using the node; 100% resources are available otherwise. We assume that on average a node is used actively for 10 hours per day by the end-user. We further assume an average upload bandwidth of 50 KBps (i.e., what Comcast cable Internet provides currently to non-commercial customers), which is quite conservative considering that other broadband options like DSL, VVD Communication [30] or Utopia [28] provide better bandwidth, and countries like South Korea and Japan have broadband connections providing Mbps of upload bandwidth. We assume that each node in the collective contributes on average 5GB of storage, which is a quite conservative estimate considering the sizes of modern hard disks. For processing capabilities, we assume a 2GHz processor with 1GB of RAM.

4.1 Collective

A million nodes with 50 KBps of upload bandwidth means an available aggregate upload bandwidth of 50 GBps. Since we assume that on average only 10% of each node's capacity is available for 10 hours each day, we can probabilistically estimate the available upload bandwidth as $(5*10+50*14)/24$, i.e., 31.25 GBps at any instant, although not constant. Similarly we can estimate that at any instant this collective will have processing capabilities worth $(0.2*10+2*14)/24*10^6$ = 1.25 * 10^6 GHz. For storage, disk space remains available all the time irrespective of the node's use by the end-user or not. Thus we estimate $5*10^6$GB of available storage.

Typically there will be an overhead in terms of bandwidth and storage to maintain service level properties. For example, bandwidth will be used to maintain caching in a content distribution system. Similarly there will be storage overhead to maintain durability (using either straightforward replication or erasure coding). Assuming 10% overhead for bandwidth, we are left with 28.125 GBps of bandwidth. Assuming 80% overhead for storage (required for 4 extra replicas), we are left with 10^6 GB of storage available.

4.2 Self Managed Cluster

Assuming dual core 2*3GHz machines, we can build a comparable computing cluster using $(1.25*10^6)/6)$ = 200K such machines. So let us use 100K machines as a conservative estimate for our analysis here. Assuming each machine costs around $1000 including storage, the initial investments for such a cluster will be around $100 million. Plus we will need to upgrade such cluster periodically to keep up with technology advancements. Assuming a 5-year life cycle, we need to depreciate it at $40M per year, i.e, $1.66M per month.

To estimate data center and bandwidth costs, we use the colocation costs advertised on websites like *creativedata.net*, *colocation.com*, and *apvio.net* as a guide. For bandwidth, the typical costs advertised on were around $45 per Mbps per month. Using that estimate, for 28.125 GBps we require $28.125*8*1000*45$ = $10.125 million per month. Typical cabinet prices for these datacenters start at $650 for 40 units. Using this as an estimate, we require $50*1000*650/40$ = $0.8125 million per month. Adding these two costs and ignoring setup fees and related costs, we require $10.93 million per month to operate.

As reflected in some of the TCO (total cost of ownership) studies available on Internet, administrative personal costs are one of the biggest part of overall cost of managing a cluster. For example, a TCO study by the hostbasket web hosting company [27] puts the labor cost at 54% of total cost, while Aruba.it [26] puts the labor cost at 41% of the total cost. We do not have any good way to quantify our costs here, so we use a conservatively estimate of $3 million per month for labor costs.

Thus overall we will need around $15.5 M per month to build a cluster equivalent to a million node collective.

4.3 Utility Computing Services from Amazon

Amazon web services [2] provides utility computing services for computing, storage, and bandwidth through Simple Storage Service (S3) and Edge Computing Service (EC2). EC2 provides an instance of 1.7Ghz x86 processor, 1.75GB of RAM, 160GB of local disk, and 250Mb/s of network bandwidth. It has following prices: $0.10 per instance-hour consumed, $0.20 per GB of data transferred into/out of Amazon (i.e., Internet traffic). S3 has the following pricing: $0.15 per GB-Month of storage used and $0.20 per GB of data transferred.

Considering the computing resources of 1.25 * 10^6 GHz, we will require around 735K instances. Let's take an conservative estimate of 300K instances for 24x30 hours; it will cost $(300*1000*0.1*24*30)$ = 21.6 million dollars per month. These machines will have enough storage space available for matching $5*10^6$ GB storage of collective. If we use S3 for storage, we will require 0.15 million dollars per month. For bandwidth, costs are same for both EC2 and S3. Based on 28.125GB/s, we will require 14.58 million dollars per month for bandwidth. This is a very conservative estimate, as amazon does not promise a bandwidth of 28.125 GBps with that pricing.

Thus overall we will require around $36M per month to have a system comparable to a million node collective.

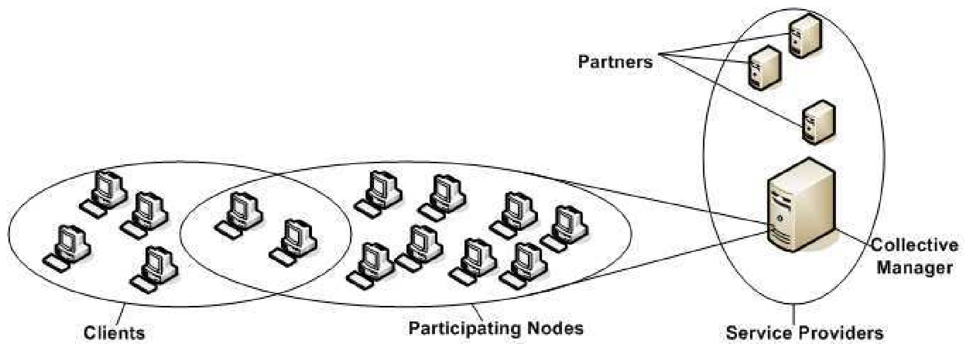

Figure 2: Main Players in Collective

4.4 Opportunity

Now that we have a quantitative idea of the cost of a collective-equivalent cluster, we can use that to get an estimate of potential rewards possible for participating nodes. So from the raw resource point of view, a collective can provide around \$15 to \$30 per month to participating nodes having resources as defined in above assumptions. From a service point of view, services built on collective overlay may be worth much more than raw resources, and hence it may be possible to give even better returns. Additionally a collective can provide return services to the PNs in addition to direct cash payout. This can increase the profit margins even more.

Overall we are trying to monetize resources (cycles, storage, and network) that have already been paid for to support something else (e.g., I already own a PC so I can surf the net; business already have machines that they use to run their business). Thus a collective can provide equivalent services at a more competitive rate than a cluster based approach, while still providing decent return back to its participating nodes. Additionally a collective does not require huge amount of initial investment that is needed for a setting up a big cluster. It does not require the costly maintenance or power costs. End-nodes are upgraded in due time by their users, thus it gets the advantage of technology advancements for free.

5 Collective Design

The collective system as a whole is a collection of distributed services built by aggregating the idle resources provided by willing end-nodes in return for compensation. There are four main players:

1. **Participating Nodes (PNs)** are end nodes that have idle compute, storage, or network resources. They are typically connected to the Internet though some sort of broadband service. These nodes have different compute/communication capabilities, go up and down in unpredictable ways, and are inherently untrusted.

2. **Collective Managers (CMs)** are service providers to whom individual nodes provide their idle resources. A CM uses these resources to provide a set of meaningful services to clients and then compensates the PNs. Multiple competing CMs can co-exist, each providing different services and/or pricing models.

3. **Clients** are individuals that wish to utilize the services offered by CMs, e.g., using a collective remote backup service or downloading a video from the collective content distribution service.

4. **Partners** are commercial users of a CM, e.g., an online movie distribution company can utilize the collective service for movie distribution, while managing the content acquisition and sales itself.

A service may not necessarily have both clients and partners. For example, a compute service may have partners but no clients, while a remote backup service may have clients but no partners. The Collective Content Distribution Service (CCDS) has both clients and partners. Figure 2 illustrates main players in a collective. PNs can also be clients of services offered by the collective overlay.

5.1 Participating Nodes

To provide resources to a CM, a PN runs a VM instance and provides root access to that instance to the CM. The decision to use a virtual machine instance as the unit of resource allocation has several important advantages over alternative approaches. Virtual machine technology provides greater *isolation*, *flexibility*, and *resource control* than simply running application processes directly on top of a standard Unix or Windows box.

In terms of isolation, applications running on a virtual machine instance cannot directly interfere with applications running on the host machine, nor can they access resources (e.g., the file system) reserved for the host machine. VM technology makes it effectively impossible for rogue client software to access resources to which it does not have access rights, install viruses, or “crack root”.

The virtual machine monitor can enforce resource controls (e.g., disk quota, cpu share, and physical memory allocation) on a per-VM basis. This design allows normal work to proceed on the host machine without undue impact by applications running on a VM. The protection and resource controls provided by VMs will make people more willing to make their machines accessible to a collective overlay, without fear that they will be misused or infected.

Using VM technology also provides advantages to a collective manager. A collective manager has `root` access and can install and execute arbitrary software on participating nodes. This design provides tremendous flexibility – what a collective manager can do is not limited by what some middleware layer supports. Programmers can use different middleware layers like CORBA, RPC, or SOAP based on their needs, which enables our system to support a wide variety of distributed services and applications. For our prototype, we use the free VMware Player [29] and Xen [3] for the VM layer.

In addition to the VM, each PN runs a small *node-agent*. The *node-agent* handles the basic interaction between the user and CM, e.g., to determine when the node has sufficient idle resources to warrant joining the CM’s collective or to start/stop the VM. The *node-agent* provides a simple UI through which the user can set limits on the resources provided to the collective (e.g., limits on disk/memory space or limits on network bandwidth that may vary depending on the time of day). The *node-agent* also provides a way for the host to temporarily suspend/resume the collective’s VM.

5.2 Collective Manager

A typical distributed service built on a collective consists of components that run colocated on the CM (called *service managers*) and other components that run on the PNs. A service manager is responsible for converting service requirements into small components and distributing these components to a set of PNs. Typically each service component will be replicated to provide availability and scalability.

As an example, Figure 1 shows how we might provide a collective content distribution service (CCDS). Here a content distribution partner collaborates with the CM to provide a content distribution service. The content distributor interfaces with the service manager to distribute (probably encrypted) content. The service manager divides the content into multiple chunks, and proactively caches them across multiple PNs.

Clients run an application, e.g., an iTunes content download application, that interacts directly with the content distributor for browsing and sales. After a client purchases a particular song or video, the content distributor sends it a signed certificate that gives the client the right to download the song/video from the

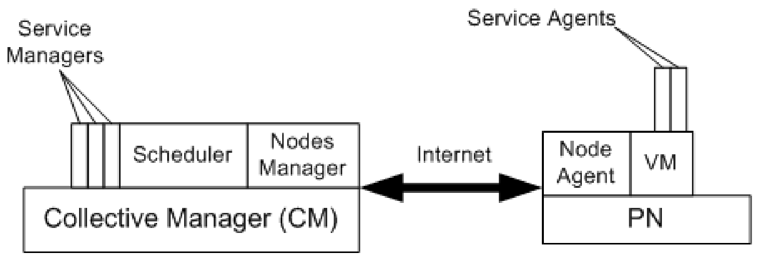

Figure 3: System Architecture

CCDS overlay network and a contact list of PNs. The client then downloads the content directly from PNs, with different chunks coming from different nodes.

Figure 3 shows the high level architecture of a collective. Apart from the service managers, the CM consists of a *node manager* and a *scheduler*. The node manager tracks the set of PNs currently registered with the CM, including their available resources and resource restrictions. The scheduler helps schedule the resources on individual PNs. A given PN can run multiple services.

5.2.1 Failure tracking and Liveness server

Node churning and failures are an inherent part of a system built from end-nodes distributed across the Internet. We use multiple methods to detect node failures (churn) in a timely fashion. The techniques we use to determine when a participating node has failed or left the collective are as follows:

- **Centralized liveness server**: Every node sends a join/leave message to a centralized liveness server whenever it joins the collective or shuts down gracefully.
- **Application-level alertness mechanism**: The CM, other PNs, and clients regularly contact other nodes as part of normal service operations. If they are unable to contact a node, they inform liveness server.

The liveness server (also called the *node manager*) keeps track of all registered nodes, as well as currently active (online) nodes in the collective. Whenever a participating node joins/rejoins the collective, it pings the *node manager* with an 'I-came-online' message. On receiving that, the node manager adds that node to the active node list. That node remains there until the node-manager receives a direct or indirect indication of node's not being online. A direct indication is sent by a node-agent if a user temporarily disables the node's participation in collective or when a node shutdowns gracefully (e.g. as part of node's shutdown). Indirect indications are reported either by service manager, clients or other PNs, whenever they happen to contact the node for certain data, and does not get any response. On direct indication, CM removes the node from the active list. Indirect indications may be a genuine shutdown or failure, but it can also result from network partitioning, or due to wrong reporting by a malicious client/PN. So on indirect indication, CM adds the list to a check-alive list. CM pings the nodes on check-alive list few times (mostly when CM is free from other work) before it removes the node from active list.

Apart from the active/non-active status, node manager keeps historical record of each node. Basically different node-agents running on PNs collect lot of useful information - e.g. observed upload bandwidth, node's boot up timings etc., and send these information to the node-manager periodically (e.g. after every 3 days).

5.2.2 Scheduler

One of the biggest challenge of CM is to effectively utilize the resources of participating nodes towards meaningful activities. CM need to understand the importance, and resourcefulness of individual nodes, and should try to maximize the utilization of its idle resources. At the same time, appropriate resources should be made available to the different services to maintain acceptable performance.

The scheduler helps schedule available resources on individual PNs to different services running on the collective. The scheduler uses historical data available at the node manager to group PNs according to their typical availability (when they typically enter/leave the collective) and resources (how much processing, storage, and network resources are available). These grouping are then used to decide on which node a given service component should be scheduled.

We can divide scheduler decision making process into two main categories - past history based scheduling and reactive scheduling.

Past History Based Scheduling

The collective incentive model rewards consistency of participating nodes and hence it leads to nodes staying for longer duration in the system. This provides an opportunity to learn about participating nodes' available resources and performance over an extended period of time. This information is used strategically during scheduling of services.

A collective manager (CM) periodically collects historical data about all participating nodes in the system. The historical data is collected with the help of a small agent called *node-agent* that runs on every node participating in a collective. A *node-agent* collects lot of useful information - e.g. observed upload bandwidth, node's boot up timings etc., and send these information to the collective manager periodically (e.g. after every 3 days). Based on these information, the collective manager has an idea about each participating node resources, and its past history about active/non-active timings in the collective. Typically the collective manager creates different clusters (information lists) based on different desired behaviors - e.g., nodes having longest active time during last 5 days, during last month, or node having highest upload bandwidth etc. These information lists are then used to make informed decisions for various activities, e.g., to decide the caching pattern of a content. Nodes can be grouped based on a node's up/down timings, a node computing or storage or networking capabilities or a node network location.

Reactive Scheduling

Apart from a proactive information based scheduling, a collective system can reactively take actions based on observed dynamic behaviors. Here we describe some of the possible approaches towards reactive scheduling.

- **Service Demand**: A collective manager can monitor service demand rate with the help of partners (e.g. content distributor in CCDS) and then use that information to achieve better scheduling of resources. For example, if a there is a rise in a particular content demand (e.g., from a sale of 10 per day to 1000 per day), the scheduler can increase replication to handle the increased load.

- **Churning Detection**: Another technique is to detect failures/churning and then take actions to mask those failures by creating more replicas when a previous replica fails.

- **Client Reports**: Another approach to get feedback is based on performance reports sent by client application. For example, if a client downloading a content does not get sufficient bandwidth, it can send a report to the collective manager. The collective manager can then take actions to fix the problem by creating more replicas or moving replicas to better nodes.

5.3 Security

The basic security problems that must be addressed in a collective infrastructure are (i) how to uniquely identify and authenticate each entity, (ii) how to ensure that a PN is not misused or corrupted as a side-effect of participating in a collective, and (iii) how to handle selfish or malicious behaviors.

5.3.1 Identity and Authentication

Each PN and each client is identified by a unique machine-generated ID and a unique public/private key pair. The CM acts as the root of the public key infrastructure (PKI) employed by its collective. Each PN and client is issued a certificate signed by the CM that associates the public key of the PN or client with their unique IDs. Similarly each partner also is identified by a unique public/private key pair. These keys and certificates are used to create secure communication channels and to digitally sign the reports sent to the CM.

5.3.2 Trust and Access Control at PNs

The collective uses a VM sandboxing environment where a PN runs a VM instance to provide resources to a CM. This ensures that the collective software is isolation from the host PN. That is, applications running inside the VM cannot directly interfere nor access resources belonging to the host PN. Additionally the host PN can enforce resource controls such as disk quota, cpu share, and physical memory allocation for the VM. This allows normal work to proceed on the host PN without undue impact. The protection and resource controls provided by VM technology will make people more willing to make their machines accessible to the collective, without fear that they will be misused or infected.

Even though VMs provides good isolation, a malicious user can still misuse the virtual machine to launch network attacks [12]. This problem is handled through access control, i.e., a VM instance on a PN can only be directly controlled by the CM. We do not allow external entities to run arbitrary code on VMs. All entities other than the CM interact with VMs only through a well defined application-level protocol.

On the flip side, a host PN has complete access to the VM running on it. A selfish PN administrator can potentially see or even modify files and data loaded on the virtual machine. Selfish behaviors and prevention mechanisms are discussed in the next section.

5.4 Incentive Model and Selfish Behaviors

Since collectives may include selfish nodes, it is important to mitigate the negative effects of selfish behavior. Selfish nodes strive to earn more than their fair share of compensation. Selfish behavior has been observed extensively in distributed systems, e.g., free riding in Gnutella [1] and software modifications to get more credit than earned in SETI@home [16].

For a collective system to work, the system must discourage dishonest behaviors (e.g., cheating users who lie about how many resources they have provided) and encourage nodes to stay in the collective for extended periods of time.

To address these challenges, we have designed an incentive system based on game theory and the economic theory behind law enforcement that motivates just these behaviors. In 1968, Becker [4] presented an economic model of criminal behavior where actors compare the expected costs and expected benefits of offending, and only commit crimes when the expected gains exceed the expected costs. Since then there has been significant research extending the work of Becker – Polinsky et. al [21] provides a comprehensive overview of the research dealing with deterrents in law enforcement. In this section we describe our incentive system and our mechanisms to discourage dishonesty.

In a collective system, a PN's compensation is based on how much its resources contribute to the success of services running on the collective. A CM shares its profits with PNs in proportion to their contribution towards different services. For example, in the CCDS example, PNs will receive a fraction of the money paid by the content distributor roughly proportional to the fraction of the total content that they deliver. The basic unit of compensation is a CM-specific credit that acts as a kind of currency. Users can convert credits to cash or use them to buy services from the CM or associated partners.

For the incentive system to work, the CM needs an accurate accounting of each PN's contribution. The CM cannot simply trust the contribution reported by each node, since selfish nodes can exaggerate their contributions. In this section we discuss how we discourage selfish behavior economically.

5.4.1 Contribution Accounting and Accountability

Contribution accounting is mostly done at the service level and depends on the design of the service involved. The basic idea is to collect information from multiple sources (e.g., PNs, partners, clients, and the CM) and do offline data analysis to decide the individual node's contribution. We employ the following mechanisms:

Credits Earned $\propto$ Work Performed: The work performed to support a service invocation, e.g., downloading a movie, should be credited to the appropriate PNs. Each PN sends a detailed daily report of its activities to the CM. In the absence of selfish/malicious PNs, each service activity can be credited to unique contributing PNs. If nodes are selfish, more than one node will request credit for the same work. To resolve conflicts, the accounting system needs additional information.

Accountability: Each PN and each client is identified by a unique public/private key pair. The CM acts as the root of the public key infrastructure (PKI) employed by its collective. Each PN and client is issued a certificate signed by the CM that associates the public key of the PN or client with their unique IDs. These keys and certificates are used to create secure communication channels and to digitally sign the reports sent to the CM.

Offline Cheater Detection: To identify selfish nodes, the system collects data from PNs, CM scheduling records, service scheduling records, partners' sales records, and even completion reports by client applications (if available). This data is used to resolve conflicts by comparing what work nodes claim they did against what other entities claim was done. Conflict resolution is done offline periodically (e.g., daily). With multiple information sources, it is possible to detect selfish behaviors by PNs. However, we do not assume that CMs will be able to detect all instances of selfish behaviors.

Collusion: Of course, PNs can collude with each other and with clients. Collusion allows cheaters to confuse the CMs by providing incorrect reports from multiple channels. We counteract this by using service-specific mechanisms to make it economically un-attractive to collude.

5.4.2 Variable Pay Rates (Raises and Cuts)

To provide an incentive for nodes to provide stable resource levels and to penalize node churn, the amount of credits received by a node in return for work depends on the node's long term *consistency*. A node that remains in the CM's pool for long periods of time and that provides continuous predictable performance receives more credit for a unit of work than a node that flits in and out of the CM's pool.

Credit-per-unit-work (pay) rates are divided into levels. PNs enter the system at the lowest pay rate; a node's pay rate increases as it demonstrates stable consistent contributions to the collective. The number of levels and the behavior required to get a "pay raise" are configurable parameters for any given service.

To discourage selfish behavior, the system can apply a pay cut when it identifies a node mis-reporting the amount of work it performs. The size of the pay cut can be configured on a per-service basis. Selfish behavior in one service leads to pay cuts in other services run on that node. As an alternative, we could ban PNs from the system when they are caught cheating, but doing so eliminates nodes who might "learn their lesson" after finding that cheating does not pay in the long run. If a node continues to cheat, its pay rate becomes negative (i.e., it accumulates debt that must be worked off before being paid), which has the same effect as simply banning them.

Other factors can be applied to determine a particular node's pay rate. For example, nodes that are particularly important to a given service due to their location or unique resources (e.g., a fat network pipe or extremely high availability) may receive a bonus pay rate to encourage them to remain part of the CM's pool.

5.4.3 Summary

Our incentive model employs a currency-based system that rewards work performed, as well as the consistency of the work. Further, it is a well known phenomenon in game theory that repeated interactions give rise to incentives that differ fundamentally from isolated interactions [19]. Thus, the collective manager employs offline analysis of data provided by participating nodes, partners, clients, and collective managers to determine future pay rates for each node. Consistently desirable behavior leads to increased rewards, e.g., the pay rate of nodes increases in response to predictable long term availability. Undesirable behavior results in decreased rewards, e.g., the pay rate of nodes decreases in response to being caught lying about work done in an attempt to receive undeserved compensation.

In [13], we analyze the impact of our incentive model from an economic standpoint to derive key properties of our incentive system. We examine the impact of decisions made by selfish nodes and analyze the gain vs loss possibilities for participating nodes as we vary the likelihood of bad actors being caught. We show that while we cannot prevent users from being dishonest, our mechanisms mitigate dishonest behavior by making it economically unattractive. We show that a small probability of catching cheaters (under 4%) is sufficient for creating a successful deterrence against selfishness. We further show that our incentive system can be used successfully to motivate nodes to remain in the system for prolonged durations.

6 Conclusion and Future Work

In this paper, we present a new *collective* model for exploiting the idle compute, storage, and networking resources of unreliable and untrusted computers distributed around the world. Unlike previous efforts to harness idle distributed resources, we propose a system based on *collective managers* that provide explicit credits for work performed on behalf of services. To discourage selfish behavior, we use a combination of offline data analysis to detect selfishness and an incentive model that encourages stable, collusion-free, unselfish behavior. Collectives provide a useful alternative to the dominant pure P2P approach; they utilize idle resources more effectively, have a more meaningful economic model, and are better suited to building legal commercially interesting services.

We believe that a collective system centered around competing CMs can grow to millions of nodes and serve as an excellent infrastructure for building new and interesting services that exploit otherwise idle resources. We are currently developing a content distribution service, a backup service, and a high-performance computing service based on collectives.

References

[1] E. Adar and B. Huberman. Free riding on gnutella. *First Monday*, 5(10), October 2000.

[2] Amazon Web Services. `http://www.amazon.com/aws/`.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th Symposium on Operating Systems Principles (SOSP)*, October, 2003.

[4] G. S. Becker. Crime and punishment: An economic approach. *The Journal of Political Economy*, 76(2):169–217, 1968.

[5] Cachelogic. P2p in 2005. `http://www.cachelogic.com/home/pages/research/p2p2005.php`.

[6] Cachelogic. True picture of file sharing, 2004. `http://www.cachelogic.com/home/pages/research/p2p2004.php`.

[7] B. Calder, A. Chien, J. Wang, and D. Yang. The entropia virtual machine for desktop grids. In *International Conference on Virtual Execution Environment*, 2005.

[8] B. Cohen. Incentives build robustness in bittorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, 2003.

[9] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid - enabling scalable virtual organization. *Internation Journal of Supercomputer Applications*, 15(3), 2001.

[10] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. Sharp: An architecture for secure resource peering. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003.

[11] Gnutella. `http://www.gnutella.com`.

[12] S. Goyal and J. Carter. Safely harnessing wide area surrogate computing -or- how to avoid building the perfect platform for network attacks. In *Proceedings of the First Workshop on Real Large Distributed Systems*, Dec. 2004.

[13] S. Goyal and J. Carter. Ensuring prolonged participation and deterring cheating behaviors in a collective. Technical Report UUCS-08-010, School of Computing, University of Utah, September 2008.

[14] S. Guha, N. Daswani, and R. Jain. An experimental study of the skype peer-to-peer voip system. In *5th International Workshop on Peer-to-Peer Systems*, Feb. 2006.

[15] Ion Stoica *et al*. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, pages 149–160, Aug. 2001.

[16] L. Kahney. Cheaters bow to peer pressure. *Wired*, 2001.

[17] Kazaa. `http://www.kazaa.com`.

[18] M. Litzkow, M. Livny, and M. Mutka. Condor — a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, June 1988.

[19] G. J. Mailath and L. Samuelson. *Repeated Games and Reputations*. Oxford University Press, 2006.

[20] PlanetLab. `http://www.planet-lab.org`.

[21] A. M. Polinsky and S. Shavell. *The Theory of Public Enforcement of Law*, volume 1 of *Handbook of Law and Economics*. North Holland, Nov 2007.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of the ACM SIGCOMM Conference*, Aug. 2001.

[23] D. Reed, I. Pratt, P. Menage, S. Early, and N. Stratford. Xenoservers: Accounted execution of untrusted code. In *IEEE Hot Topics in Operating Systems (HotOS) VII*, Mar. 1999.

[24] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *International Conference on Distributed Systems Platforms*, Nov. 2001.

[25] SETI@home. `http://setiathome.ssl.berkeley.edu`.

[26] M. S. P. S. C. Study. Aruba.it. `http://download.microsoft.com/download/6/b/e/6be5466b-51a5-4eaf-a7fc-590f32bc9cb3/Aruba.it%20Case%20Study.doc`.

[27] M. S. P. S. C. Study. Hostbasket. `http://download.microsoft.com/download/b/f/3/bf34b7be-81e9-46a8-a5e3-ccb648a98547/Hostbasket%20Final.doc`.

[28] Utopia. `http://www.utopianet.org/`.

[29] VMware Player. `http://www.vmware.com/player`.

[30] VVD Communications. `http://www.vvdcommunications.com`.

[31] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, January 2004.