P2P Resource Pool and Its Application to Optimize Wide-Area Application Level Multicasting

Zheng Zhang, Yu Chen, Shi-Ding Lin, Bo-Ying Lu, Shu-Ming Shi*, Xing Xie and Chun Yuan
Microsoft Research Asia
5F, Sigma building, No.49, Zhichun Road
Beijing, 100080, P.R.China

{zzhang, ychen, i-slin, t-bylu, xingx, cyuan}@microsoft.com
ssm01@mails.tsinghua.edu.cn

ABSTRACT

The concept of resource pool has a very long history. Propelled by the need to share CPU cycles of supercomputers for high-throughput computing jobs from the scientific community, the vision is most recently explored by the advocates of Grid. On the other hand, the advent of P2P researches has demonstrated the feasibility of integrating potentially unlimited amount of less powerful machines around the world. Organizing a P2P resource pool thus becomes an interesting research topic.

This paper attempts to address two problems. The first is how to organize a P2P resource pool, and our answer is to combine the self-organizing strength of P2P DHT with an in-system, self-scaling monitoring infrastructure that is layered on top of DHT. The second question is the utility of the P2P resource pool for interesting applications. And we choose to showcase its power by optimizing wide-area application level multicasting (ALM), a problem far more challenging and interesting than conventional tasks such as massively parallel computation.

We show that utilizing spare resources in the pool results in significant savings for single ALM session. Furthermore, we adopt a purely market-driven approach to optimize multiple concurrent sessions. As expected, sessions of higher priority are given higher share of resources.

1. INTRODUCTION

The concept of resource pool has a very long history. Most recently, the Grid community has propelled this vision by uniting the local resource pools (i.e. cluster) spread over a dozen sites into a global one. However, the applications are restricted mostly in the area of high-throughput computing.

The advent of P2P paints a different direction. First of all, the research community has been actively investigating applications beyond that of number crunching. The scale and composition of resources are of an entirely different nature: we are talking about millions of desktop PCs spread widely apart. If a P2P resource pool is attempted, it is not clear that the same technologies in Grid can be adopted in a straightforward manner.

On the other hand, A P2P resource pool differs from typical P2P *applications* in that there can be multiple active instances of different applications running in the pool. Also, a peer in the pool may be helping an application instance of which it is not necessarily a member.

Broadly speaking, we define a *P2P resource pool* as a collection of desktop-grade resources on the edge of the internet, and some form of middleware is managing these resources such that they can offer computing, storage and networking capabilities for multiple instances of potentially different distributed applications.

This paper addresses two related and important questions with respect to P2P resource pool: its architecture and utility to schedule interesting applications. We propose to employ structured P2P in the form of DHT (distributed hash table) [25][30][36][42] to pool together potentially unlimited amount of and widely distributed resources. However, DHT alone does not automatically generate a resource pool. Thus, we combine the self-organizing capability of P2P DHT with an in-system, self-scaling monitoring infrastructure SOMO (*Self-Organized Metadata Overlay*). SOMO builds a dynamic system status database which is available internally to any peers. This database is being continuously updated and thus creates an illusion of a single, large resource pool.

Next, we demonstrate the utility of P2P resource pool by describing how to schedule and optimize application-level multicasting (ALM), an application that is far more challenging than, for instance, massively parallel computations. By using peers that are otherwise idle, our finding is that up to 30% improvement can be made for small-to-medium group size. In the context of scheduling multiple competing sessions, our approach is remarkably simple: global scheduling is *never* attempted. Instead, we adopt a market-driven approach and let each session competes based on their priorities, armed with the information provided by SOMO. Our results show that, as expected, sessions of higher priority are given higher share of resources, resulting in better performance.

The rest of the paper is organized as follows. We articulate the need of a P2P resource pool in Section-2 and contrast it with existing alternatives. The architecture of a P2P resource pool is described in Section-3. For many interesting P2P applications, there is a need to generate resource attributes that can not be derived locally from a machine. For instance, network coordinates and bottleneck bandwidth are necessary to evaluate potential helping peers in ALM. This problem is addressed in Section-4. Section-5 evaluates how P2P resource can help ALM and provides experiment results. We discuss related work in Section 6 and conclude in Section 7.

2. MOTIVATION AND ALTERNATIVES

2.1 The argument for a P2P resource pool

Over the recent years, the P2P research community has investigated many interesting P2P applications; ranging from wide-area distributed storage [20][9][10], scientific computing [18], application-level multicasting [6][4], distributed web cache [35], searching [29] and collaborative spam fighting [43], to name just a few. There is, however, a common assumption that underlies all these proposals: all peers are active participants in *one* application instance.

* This work was performed when the author was a part-time student at Microsoft Research Asia.

P2P resource pool explores a different dimension in which 1) there can be multiple and simultaneous instances of different applications and they could potentially overlap on the resources they are running; and 2) a peer maybe helping an application instance of which it is not a member. The first point illustrates aggregated power of all potential resources, and the second reflects and extends the very collaborative principle upon which the P2P premise is found.

For instance, all existing application-level multicasting (ALM) algorithms assume that the only resources available are those in the ALM session. In a collaborative environment, many other stand-by resources could be included for an otherwise more optimal solution. For example, Microsoft Research has five branches across the globe, and has many thousands of machines that are geographically distributed. At a given hour, however, number of active video-conference sessions is likely to be only a handful, and each session may have a small number of participants (say less than 20).

The availability of a P2P resource pool offers new optimization possibilities. As shown in Figure 1, when an otherwise idle but suitable helping peer is identified, it can be integrated into a topology with better performance. This is an actual output of our algorithm used in this paper.

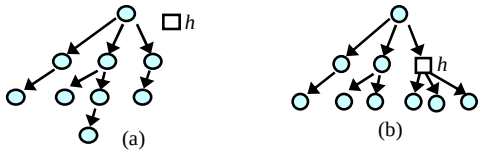


Figure 1. (a) An optimal plan for an ALM. (b) An even better plan using helper nodes in the resource pool. Circles are the original members of the session, and the square is an available peer with a large degree.

The P2P resource pool has seen its first incarnation in PlanetLab [24], a wide-area P2P testbed. On PlanetLab, researchers upload experiments on to machines comprising the testbed that will be run concurrently with others. Up-to-date, its scale is still limited (220 nodes as of the writing of the paper), and thus a need to organize the pool in a more scalable and self-organizing fashion is not yet profound. This is one of the problem this paper attempts to address.

2.2 Resource pool and its alternatives

We wish to give a more concrete definition of resource pool by contrasting it against another interesting alternative: the *job pool*. This is necessary because, from a high-level perspective, both are venues to deliver the matchmaking between job and resource, and that neither is perfect: depending on the application scenario, each has its unique strength and weakness.

Informally speaking, a job pool is a collection of jobs and is where an idle resource look for suitable work to perform, whereas a resource pool is the precisely the opposite: a task manager goes into a resource pool to discover and acquire necessary helping hands in order to accomplish a given mission. Of course, in a distributed environment, there can be legitimate combination of both.

A perfect example of a p2p job pool is SETI@home [18] (and of course, many others of the same flavor). There is a well-maintained central site and, typically, the application should be easily parceled out for distribution. Machines register themselves in order to grab a piece of work and then go away cranking away whenever they feel like. This is a very economical model and requires very limited amount of management at the central site. Provided that job is of coarse granularity, a centralized architecture works extremely well. It has been reported that SETI@home has aggregated computing power far exceeding some of the most powerful supercomputers in the world. The limitation is also obvious. Although it is possible to think of advanced variations, because the unpredictability of when and what resources will become available, applications are restricted mostly to those that are conventionally known as "embarrassingly parallel" ones. It is also possible to implement the job pool as a distributed architecture, but it will be far easier to just use a centralized architecture.

In contrast, a node joins a resource pool in the hope that its power, when otherwise idle, can be of some use. The economic incentive can be stronger, especially in the context of P2P: tasks of arbitrary type (beyond those of number crunching) will tap into the power of other participants in the pool at some suitable point. The added flexibility is particularly useful for applications such as running application-level multicasting sessions with some level of QoS guarantee. This is so because planning the topology of the tree is itself a complex piece of work. On the other hand, the consequences are many. Foremost of all is an accurate accounting of what is going on in the resource pool. This is necessary for each job to quickly query the available candidates and subsequently make resource reservation. The implication is that, given the potentially huge amount of resources in the millions, a client-server architecture where each client updates *one* central entity about its status is no longer a scalable – not to say robust alternative.

To summarize, job pool is best for scenarios where the task can be well-partitioned. Resource pool can ideally accommodate tasks of arbitrary type. However, it will need, as a minimum, a scalable way of monitoring and aggregating system information so that resource reservations can be carried out at the discretion of task managers that is responsible for individual incoming jobs.

The principle of resource pool is what motivates the work in the Grid space, in particular the Condor-G line of work [14][8]. For instance, the Grip Resource Registration Protocol (GRRP) is used for an entity, typically representing a cluster of machines, to notify other entities that it is part of the pool. Grip Resource Information Protocol (GRIP), on the other hand, is the primitive to construct aggregated resource directory service through which tasks can query for potential candidates. The Condor-G agent can use such infrastructure to submit jobs and monitor its progress.

There have been many discussions about the convergence of P2P and Grid [13][22]. We believe that indeed there are many synergies among the two in the space of resource pool organization. In particular, we argue that the self-organizing attributes is what the many excellent work of P2P can bring to the scene of Grid. We will offer a more elaborate discussion at the conclusion of Section-3.

3. BUILDING P2P RESOURCE POOL

The foundation of our resource pool proposal is the so-called *structured* peer-to-peer systems, and in particular the *distributed hash table* (DHT). DHT offers a way to pool together potentially unlimited amount of resources together. But the capacity to pool

resources together does not mean a resource pool is established; the latter requires the availability of an infrastructure to know what is going on. In this section, we will describe these points in turn, and will conclude with the comparison of Grid-based resource pool.

3.1 P2P DHT

We assume that the readers are reasonably familiar with the concept of DHT, and for the sake of brevity will only go through the basics. In DHT, a very large logical space (e.g. 160-bits) is assumed. Nodes join this space with random IDs and thus partition the spaced uniformly. The ID can be, for instance, MD5 over a node's IP address. An ordered set of nodes, in turn, allows a node's responsible *zone* to be strictly defined. Let p and q be a node x's predecessor and successor, respectively. One definition of a node's zone is simply the space between the ID of its immediate predecessor ID (non-inclusive) and its own ID. In other words: $zone(x) \equiv (ID(p), ID(x)]$. This is essentially how consistent hashing assigns zones to DHT nodes [36] (Figure 2). To harden the ring against system dynamism, each node records r neighbors to each side in the rudimentary routing table that is commonly known as *leaf-set*. Neighbors exchange heartbeats to keep their routing tables current, updating their routing tables when node join/leave events occur. This base ring is the simplest P2P DHT.

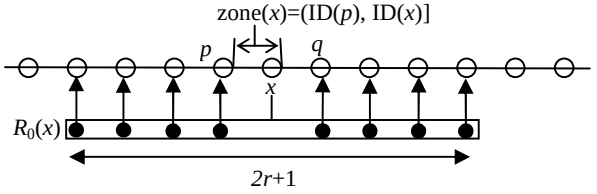


Figure 2. The simplest P2P DHT – a ring, the *zone* and the basic routing table that records *r* neighbors to each side.

If one imagines the zone being a hash bucket in an ordinary hash table, then the ring is a *distributed hash table*. Given a key in the space, one can always resolve which node is being responsible. The lookup performance is $O(N)$ in this simple ring structure, where N is the number of nodes in the system.

Elaborate algorithms built upon the above concept achieves $O(\log N)$ performance with either $O(\log N)$ or even constant states (i.e. the routing table entries). Representative systems including Chord[36], CAN[25], Pastry[30] and Tapestry[42].

The most interesting aspect of a DHT is that the whole system is self-organizing with very low overhead – typically in the order of $O(\log N)$. The second significant attribute is the virtualization of a space where both resources and other entities (such as documents stored in DHT) live together; this feature is what we explore the most in this paper.

3.2 SOMO: Self-Organized Metadata Overlay

As we described earlier, to complete a P2P resource pool, we must augment DHT with an *in-system* monitoring infrastructure. We emphasize the in-system property because for a large system, it is impractical to rely on external monitoring service. Such an infrastructure must satisfy a few key properties: *self-organizing* at the same scale as the hosting DHT, fully *distributed* and *self-healing*, and be as *accurate* as possible of the metadata gathered and disseminated. Our proposal, SOMO (for Self-Organized Metadata Overlay), was designed for this purpose and was first introduced in [41]. Its construction took a top-down approach; we describe in this paper an improved bottom-up version.

The basic idea of SOMO is to draw a logical tree with a fixed fan-out (e.g. 8) first. The positions of the tree nodes can be calculated by each brick independently. Given its responsible zone in the DHT, each node selects the highest logical tree node that it hosts as its representation in the SOMO hierarchy, and then calculates the position of the parent logical node, routes to that parent tree node to form a child-parent link. A hierarchy is thus built in a self-organized fashion. The SOMO hierarchy is completely self-govern and self-repair, and can gather and disseminate metadata in $O(\log_{fan\_out} N)$ time. Given a data reporting interval T, information is gathered from the SOMO leaves and flows to its root with a maximum delay of $\log_k N \cdot T$. This bound is derived when flow between hierarchies of SOMO is completely unsynchronized. If upper SOMO nodes' call for reports immediately triggers the similar actions of their children, then the latency can be reduced to $T+t_{hop}\cdot\log_k N$, where t_{hop} is average latency of a trip in the hosting DHT. The unsynchronized flow has latency bound of $\log_k N \cdot T$, whereas the synchronized version will be bounded by T in practice (e.g. 5 minutes). Note that $O(t_{hop} \cdot \log_k N)$ is the absolute lower bound. For 2M nodes and with k=8 and a typical latency of 200ms per DHT hop, the SOMO root will have a global view with a lag of 1.6s.

It would seem that SOMO derived all its self-organizing and self-healing property by trading off configurability. For instance, what if we want the most capable machine to be at the top of the machine? The trick is to relax the requirement that node ID is purely random. What we need to do is to make an upward merge-sort through SOMO and first identify the most capable node. This node then exchange ID with the one who currently possesses the root logical point of SOMO (i.e 0.5 of the total space [0, 1]), and this effectively changes the machine that acts as the root *without* disturbing any other peers. Note that this self-optimizing property is impossible unless we operate in the logical spaced first.

We have implemented a SOMO-based global performance monitor called LiquidEye with which we monitor over 100 machines in our lab on a daily basis. This tool employs SOMO built over a very simple ring-like DHT, and SOMO gathers data from various performance counter on each machine. The complete system status is obtained by querying the SOMO root report through a unified UI interface. We tested the SOMO stability by unplugging cables of servers being monitored, and each time the global view is regenerated after a short jitter. The reporting cycle is 5 seconds, and the leaf SOMO report is 40bytes. In a wide-area and large-scale deployment, we will opt for a less aggressive interval and also employ compression optimization. Likewise, redundant links should be added to increase the robustness; this can be easily accomplished by letting the representative virtual node to connect to a random set of parent siblings.

3.3 Discussion

To summarize, our P2P resource pool is composed of two ingredients, see Figure 3:

- **DHT**. A DHT is used not in the sense of sharing contents, but rather as an efficient way to pool together a very large amount of resources, with zero administration overhead and no scalability bottleneck.

- **SOMO**. SOMO is a self-organizing "news broadcast" hierarchy layered over DHT. Aggregating resource status in O(log*N*) time then creates the illusion of a single resource pool.

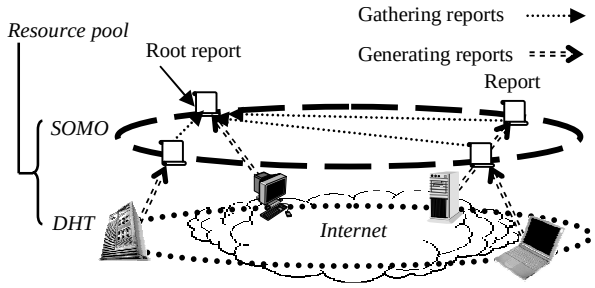


Figure 3. Combining DHT's capability of pooling resource with SOMO collectively makes the resource pool.

We are now ready to contrast the differences of a P2P resource pool versus the one created by Grid and identify their synergy. From a high-level point of view, the necessary steps are the same: register the resource, gather statistics and then aggregate them into a snapshot and, finally, ensure the resulting dynamic database can be queried by applications.

While Grid is composed mostly by dozen of sites each maybe composed by supercomputers (possibly in the form of clusters), a P2P resource pool can potentially possess millions of machines that are widely distributed. KaZaA [17], for instance, has current installation base well over 4 million.

The scale and composition of P2P resources requires that every layer be completely self-organizing, self-scaling and self-healing. There should be very little administration overhead, if at all. We have illustrated that how SOMO layered over P2P DHT accomplishes the goal. In essence, SOMO builds a skeleton upon which the aggregation is done. It makes perfect sense to adopt components of GRIP and GRRP from Grid protocols as primitives to do the job of pair-wise registration and aggregation.

4. GENERATING RESOURCE METRICS FOR ALM

For many P2P applications, resource statistics including not only the usual suspects such as CPU loads and network activities, but also more complex ones that can not be derived locally from the machine. A case in point is ALM, suppose we want to schedule a session and we obtained a huge list of potential helping peers by querying SOMO, we must select one that is nearby and also has adequate bandwidth. If only the peers IP addresses are given, pinging over them to find their vicinity is both time-consuming and error-prone.

These two critical metrics are the focus of this section. We will explain how these attributes can be generated by leveraging the interactions among DHT nodes that maintain the logical space integrity. For a more detailed treatment, we refer reader to our technical report [39].

4.1 Node coordinate estimation

The idea of coordinates-based latency estimation is proposed in GNP [12]. The point is that, in order to know *latency*(*x*, *y*), it is sufficient to compute *distance*(coord(*x*), coord(*y*)), where coord is a network coordinate in a *d*-dimension Euclidean space.

The GNP proposal relies on a set of well-distributed nodes called the *landmarks*. GNP starts by computing coordinates of the landmarks, solving their coordinates as an optimization problem. Later, other nodes compute their coordinates using essentially the same principle, but this time against the landmark nodes.

The requirement of landmark nodes contradicts to the fully distributed nature of P2P resource pool. Our observation, which is the same as Lighthouse [23] and PIC [3], is that the infrastructure nodes are not necessary. Recall that each node must heartbeats with its leafset nodes in order to collectively maintain the DHT space. If each node randomly chooses to acknowledge the heartbeat message from nodes in its leafset, then over time it will have a measured delay vector, d_m, to its leafset neighbors. In the heartbeat message, each node also reports its current coordinates. Thus, a predicted delay vector d_p is available locally as well. Node x update its own coordinates by executing downhill simplex algorithm, minimizing the function:

$$E(x) = \sum_{i=1,r} | d_p(i) - d_m(i) |$$

Notice that the optimization is done locally and only updates x's own coordinate, which will be distributed to x's leafset neighbors in subsequent heartbeats.

The above procedure is executed by all nodes periodically, and node coordinates, the measured and predicted delay vectors are being updated continuously.

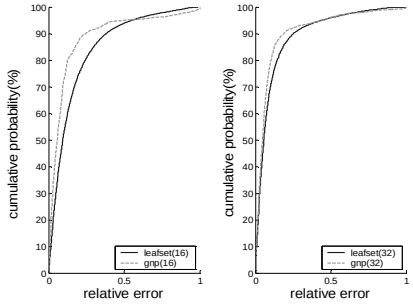


Figure 4. **CDF of relative error of GNP and the leafset-based variance; number of infrastructure node/leafset size is 16 (left graph) and 32 (right graph)**

Figure 4 presents CDF of the relative errors of the original GNP and its leafset-based version with our simulation of 1200 nodes. These 1200 nodes are distributed according to the GT-ITM [38]. As can be seen, GNP is less sensitive to number of infrastructure nodes than the leafset version is to the leafset size. Our data indicates that when L (the leafset size) is 32, which is the default configuration of Pastry, the result is very close to GNP with 16 infrastructure nodes.

4.2 Bottleneck bandwidth estimation

Network bandwidth of a peer is another important metrics for many applications running on top of P2P resource pool. It is shown that there is considerable correlation between bottleneck bandwidth and throughput [11]. Therefore, we choose bottleneck bandwidth as the predictor for throughput.

We will adopt the common assumption that bottleneck link lies in the last hop. For each node, its upstream bottleneck bandwidth is

estimated as the maximum of the measured bottleneck bandwidths from the node to its leafset members, which are limited by both the node's uplink bandwidth and the downlink bandwidths of the leafset nodes. The basic idea is that if there is one neighbor with downlink bandwidth greater than the node's uplink bandwidth, the estimation is accurate. So with more leafset nodes the chance of getting accurate estimation would be better. For the same reason, the node's downstream bottleneck bandwidth is estimated as the maximum of the measured bottleneck bandwidths from its leafset nodes to itself.

Measuring bottleneck bandwidth is well understood. For instance, in packet-pair technique [21] two packets of size S are sent back-to-back from a source node. The receiver measures the time dispersion T in between and estimates the bottleneck bandwidth from the source as S/T.

The cooperation of leafset nodes over heartbeats enables packet-pair technique to be naturally deployed. Periodically, a node x chooses to send a neighbor y two consecutive heartbeat messages back to back, padding each so that their size is sufficiently large (say 1.5KB). y now has the estimation of the bottleneck bandwidth on the path from x to itself. This value will be piggybacked in the next heartbeat to x. Likewise, y does the same probing as x. After x collects enough measured bandwidths from its leafset members, it can now estimate its own bottleneck bandwidth as above.

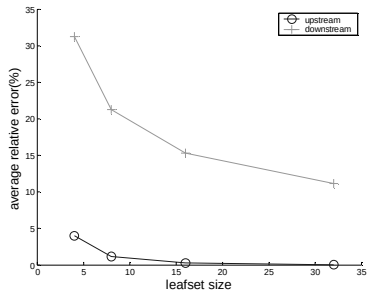


Figure 5. Average relative error vs. leafset size

We have evaluated our protocol on the Gnutella trace[1][32]. The larger the leafset, the better the prediction will be, simply because nodes with higher uplink/downlink bandwidth have a better chance to be included. This is evident in Figure 5: the average relative error (ratio of the absolute error to the real bandwidth) decreases with leafset size. Specifically, with leafset of size 32, the average relative error of upstream bandwidth estimation is almost 0 and the ranking is 100% correct. Therefore our scheme is rather effective in a practical setting. Note that uplink estimation is predicted more accurately than downlink. This is because most hosts in the Gnutella trace have downstream bandwidths higher than the upstream bandwidths of most others [32], whereas downlink can be underestimated. Fortunately, uplink bandwidth is also a more important metric in many applications.

[1] We wish to thank Stefan Saroiu and Steven Gribble for allowing us to use their traces.

5. SCHEDULING ALM SESSIONS WITHIN THE P2P RESOURCE POOL

In this section, we demonstrate how to utilize P2P resource pool to optimize for multiple simultaneous ALM sessions. The end goal is for active sessions to achieve optimal performance with *all* available and adequate peers in the resource pool. Session's performance metrics is determined by certain QoS definitions. Moreover, higher priority sessions should proportionally acquire more shares of resources. Our emphasis is for small-to-medium session size where we believe QoS is often a requirement (e.g. video-conference). We also assume static membership and denote the original set of participants as $M(s)$ for a given session s, though the algorithm can be extended to accommodate dynamic membership as well.

From a high-level point of view, the procedure is simple. The task manager of a session is responsible to run a modified heuristic algorithm to plan the topology of the ALM. This algorithm was originally developed using $M(s)$ only [34]. To utilize spare resources in the pool, the task manager queries SOMO to obtain a list of candidates. The item of the list includes not only the resource availability, but also its network coordinate as well as its bandwidth. When the plan is drawn, the task manager goes out to contact the helping peers to reserve their usages. Competing tasks will resolve their contention purely by their priorities.

We will give our QoS definition and then describe our approaches in steps. First, we will show how additional resources are recruited assuming only one single session is active. Next, we will present our approach of how multiple sessions with different priorities are optimized.

5.1 ALM QoS definition

For ALM, there exist several different criteria for optimization, like bandwidth bottleneck, maximal latency or variance of latencies. In this paper, we choose maximal latency of all members as the main objective of tree building algorithms since it can greatly affect the perception of end users. Similar to many previous works [33][34][40], each node has a bound on the number of communication sessions it can handle, which we call *degree*. This may due to the limited access bandwidth or workload of end systems.

Our definition of QoS for one given session is the same as proposed in AMCast [34] and can be formally stated as follows:

Definition 1. Degree-bounded, minimal height tree problem (DB-MHT). Given an undirected complete graph $G(V,E)$, a degree bound $d_{bound}(v)$ for each $v \in V$, a latency function $l(e)$ for each edge $e \in E$. Find a spanning tree T of G such that for each $v \in T$, degree of v satisfies $d(v) \leq d_{bound}(v)$ and the height of T (measured as aggregated latency from the root) is minimized.

Using the resource pool, the above definition is slightly extended. An extended set of helper nodes H is added to the graph, and our objective is to achieve the best solution relative to an optimal plan derived *without* using H, by adding the least amount of helper nodes.

5.2 Scheduling a single session

To make the illustration simple, we will start by describing how to schedule and optimize one single ALM session. Optimizing DB-MHT generally known as NP-complete. Our goal is not to propose

new algorithms. Instead, we will select from a few well-known ones and investigate how much performance benefits we can achieve when the resource pool is utilized.

Our base algorithm uses the one proposed in [34], with $O(N^3)$ performance bound. This algorithm can generate a solution for hundred of nodes in less than one second (Figure 6, without the code in the dashed box). This algorithm, which we refer to as "AMCast," is a typical greedy algorithm. It starts first with the root and adds it to a set of the current solution. Next, the minimum heights of the rest of the nodes are calculated by finding their closest potential parents in the solution set, subject to degree constraints. This loops back by absorbing the node with the lowest height into the solution. The process continues until all nodes are finally included in the resulting tree. To ensure that we get the best possible tree to start with, we augment this algorithm with a set of further tuning[2] (tagged as *adjust* in the rest of the paper.

```
ALM(r, V) {                 // V==M(s), r is the root
  for all v∈V               // initialization
    height(v)=l(r, v); parent(v)=r
  T = (W={r}, Link={})

  while (W<V) {             // loop until finish
    find u∈{V-W} s.t. height(u) is minimum
    if (d(parent(u))==d_bound(parent(u)-1)
      h=find_helper(u)
    if h≠NULL {             // integrate the helper node
      W+={h}; Link+={h, parent(u)};
      W+={u}; Link+={u, h};
    } else
      W+={u}; Link+={u, parent(u)};

    for all v∈{V-W} {  // re-adjust the height
      height(r)=∞
      for all w∈W
        if d(w)<d_bound(w) && height(v)>height(w)+l(w, v)
          height(v)=height(w)+l(w, v); parent(v)=w
    }
  }
  adjust(T)
  return T
}
```

Figure 6. The AMCast algorithm (without the lines in the dashed box) and the critical-node algorithm that utilizes additional helper node.

Our algorithm searching for beneficial helper nodes include two considerations: the time to trigger the search and the criteria to judge an addition. The general mechanism is described by the pseudo-code in the dash-box of Figure 6. Let u be the node that the AMCast algorithm is about to add to the solution and $parent(u)$ be its parent. When $parent(u)$'s free degree is reduced to one, we trigger the search for an additional node h. If such h exists in the resource pool, then h becomes u's parent instead and replaces u to be the child of the original $parent(u)$. Different versions vary only on the selection criteria of h but we refer to this class of optimization the *critical node* algorithm. "Critical" here means that, for a particular node, this is the last opportunity to improve upon the original greedy algorithm.

We have experimented with different algorithm to search for h. The first variation is simply to find an additional node closest to the parent node and with an adequate degree (we use 4). Let $l(a, b)$ be latency between two arbitrary nodes a and b. We find the following heuristic yields even better results:

$l(h, parent(u))+\max(l(h, v))$ is minimum
where v satisfies $parent(v) = parent(u)$ && \\ condition 1
$d_{bound}(h) \geq 4$ && \\ condition 2
$l(h, parent(u))<R$ \\ condition 3

Here, v maybe one of u's siblings. The idea here is that since all such v will potentially be h's future children, $l(h, parent(u)) + \max(l(h, v))$ is most likely to affect the potential tree height after h's joining (condition 1). Such helper node should have adequate degree (condition 2). Finally, to avoid "junk" nodes that are far away even though their degrees are high, we impose a radius R: h must lie within R away from $parent(u)$ (condition 3).

The input parameters necessary to execute the procedure include the network coordinates so that we can calculate latency between *arbitrary pair*, as well as the degrees of each node. This is made available by letting each node to publish their network coordinates as well as bandwidth constraints (using the procedure outlined in Section-4) in their reports to SOMO (Figure 7).

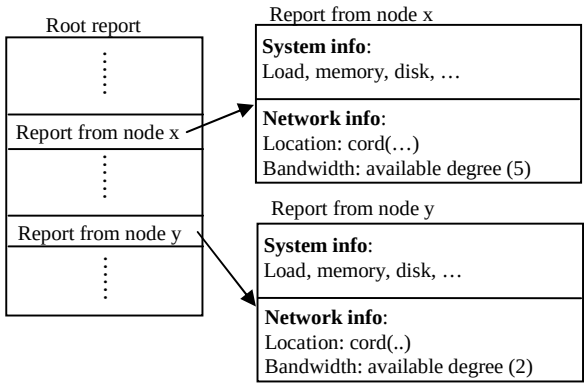


Figure 7. SOMO report structure for scheduling a single ALM session.

Our experiments simulate a two-layer Transit Stub topology [38] with 600 routers. The network consists of 24 transit routers and 576 stub routers. We assign link latencies of 100ms for intra-transit domain links, 25ms for stub-transit links and 10ms for intra-stub domain links. We also append 1200 end systems to the stub routers randomly and set the last hop latency to a random value between 3ms and 8ms. The resource pool contains all the 1200 end nodes in the network. The degree bound for all the nodes lie within 2 and 9, and follows the distribution 2^{-i} for degree i-1 when i<9 and 2^{-7} for degree 9. Thus, half of the nodes in the system have degree 2 and the population for higher degree decreases exponentially.

We found that R between 50~150 yields satisfactory results for the topology parameters we chose. The tradeoff here is that a small R will reduce the choice of candidates, whereas a larger R will introduce links of long latency in the tree. That the setting of radius to be medium range gives good result isn't a surprise. Recall that we have 100, 25, and 10 for intra-transit, stub-transit

[2] A known technique to approximate globally optimal algorithm is to adjust the tree with a set of heuristic moves. These moves include: (a) find a new parent for the highest node; (b) swap the highest node with another leaf node; (c) swap the sub-tree whose root is the parent of the highest node with another sub-tree.

and intra-stub links respectively. Thus, a radius of 50-150 will avoid all nodes from another stub.

In the followings, we call the algorithm where pair-wise node latency is known a priori via an oracle the *Critical*, and the one used the leafset estimation for vicinity judgment the *Leafset*.

We are now ready to present our results. A fair evaluation should compare our results against those of a globally optimal algorithm. Since this is not available, we report our results in terms of percentage of tree height improvement relative to the AMCast algorithm. In other words, if H_{alg} is the tree height achieved using *alg*, then:

$$Improvement = (H_{AMCast} - H_{alg})/H_{AMCast}$$

The upper bound is the latency between the furthest node to the root, corresponding to the ideal performance if the root has degree of infinity. For the data set that we used, the upper bound is between 40~50%. The average performance of these algorithms over 20 runs is shown in Figure 8 for various group sizes. It is conclusive that resource pool is very effective for small-to-medium group size. For larger groups, the original AMCast has more rooms to optimize by using the existing members already. We believe that in reality, small groups are in fact more common.

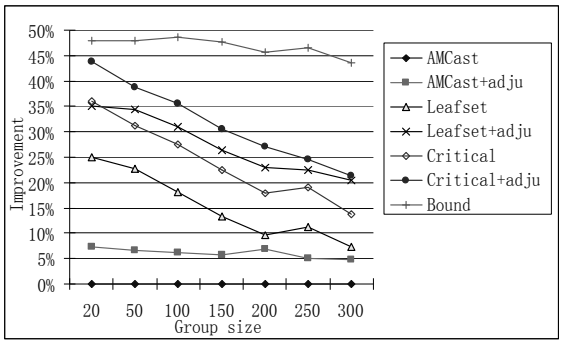


Figure 8. The performance of scheduling single ALM session. *AMCast* represents the original algorithm, *Critical* is our modified "critical node" heuristic, *Leafset* stands for the landmark based approach, *Bound* denotes the theoretical upper bound. *adju* denotes the combination when tree adjustment is performed.

For instance, *Leafset+adjustment*, which is a practical algorithm, delivers more than 30% latency reduction over the baseline for group size of 100; for group size of twenty, the reduction is 35%. Interestingly enough, tree adjustment, which is otherwise mediocre in shortening the tree (5% over baseline), is remarkably effective especially for *Leafset*. Part of the reason may be due to the inaccuracy introduced by using delay vectors to estimate node proximity.

5.3 Optimize Multiple ALM Sessions

The preceding section describes the stand-alone scheduling algorithm for one ALM session; we now discuss how to deal with multiple active sessions. Our goals are: 1) higher priority sessions are proportionally assigned with more resources, and 2) that the utilization of the resource pool as a whole is maximized.

All the sessions may start and end at random times. Each session has an integer valued priority between 1 and 3. Priority 1 session is the highest class. The number of maximum simultaneous sessions varies from 10 to 60 and each session has non-overlapping member set of size 20. Thus, when there are 60 active sessions, *all* nodes will belong to at least one session. That is, the fraction of *original* members of active sessions varies from 17% to 100%. Counting helper nodes, a session typically employ more than the original members. Also, nodes with larger degrees may be involved in more than one session.

The principle underlying our approach is very simple, and it draws insight from a well-organized society: as long as global, on-time and trusted knowledge is available, it may be best to leave each task to compete resources with their own credentials (i.e., the priorities). This purely market-driven model allows us to accomplish our goal, without the need of global scheduler of any sort.

Setting the appropriate priorities at nodes involved in a session takes extra consideration. In a collaborative P2P environment, if a node needs to run a job which includes itself as a member, it is fair to have that job be of highest priority in that node. Therefore, for a session *s* with priority *L*, it has the highest priority (i.e. 1 in our experiment) for nodes in *M*(*s*), and *L* elsewhere (i.e., for any helper nodes lie outside *M*(*s*)). This ensures that each session can be run, with a lower bound corresponding to the *AMCast+adju* algorithm. The upper bound is obtained assuming *s* is the only session in the system (i.e., *Leafset+adju*).

As before, the root of an ALM session is the task manager, which performs the planning and scheduling of the tree topology. Each session uses the *Leafset+adjustment* algorithm to schedule completely on its own, based on system resource information provided by SOMO. For a session with priority *L*, any resources that are occupied by tasks with lower priorities than *L* are considered available for its use. Likewise, when an active session loses a resource in its current plan, it will need to perform scheduling again. Each session will also rerun scheduling periodically to examine if a better plan, using recently freed resources, is better than the current one and switch to it if so.

To facilitate SOMO to gather and disseminate resource information so as to aid the planning of each task manager, as before each node publishes its information such as network coordinates in its report to SOMO. However, its degree is broken down into priorities taken by active sessions. This is summarized in the *degree table*.

$d_{bound}(x)$	4
x.dt[1]	$2(S_4)$
x.dt[2]	0
x.dt[3]	$1(S_{1\,2})$

x's degree table

$d_{bound}(y)$	2
y.dt[1]	$2(S_5)$
y.dt[2]	0
y.dt[3]	0

y's degree table

Figure 9. Two example degree tables.

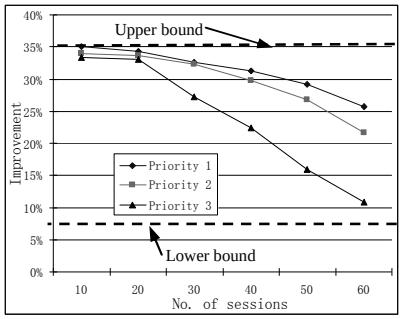
In Figure 9, we show the degree tables of two nodes. *x*'s total degree is 4, and is taken by session s_4 for 2 degrees, and s_{12} by another one degree, leaving *x* with one free degree. *y* on the other hand, has only two degrees and both of them are taken by session s_5. The degree tables are updated whenever the scheduling happens that affect a node's degree partition. Degree tables, as mentioned earlier, are gathered through SOMO and made available for any running task to query.

Ideally, the performance improvement should have a lower bound of *AMCast+adjust* where only the original member set is involved, and an upper bound of *Leafset+adjust*, when the session is the only active one in the resource pool. Therefore, performance will

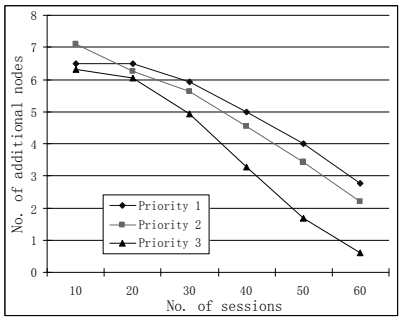
lie within 7%~35% reductions over *AMCast* (see data in Figure 8 when group size is 20).

The result is shown in Figure 10-(a). The x-axis is the number of active sessions, while the y-axis is the performance improvement. To ease the comparison, the upper bound and lower bound are also shown.

As expected, the data perfectly drop into the interval between lower bound and upper bound. When there are more sessions and overall resource becomes scarce, performance decreases across the board. However, higher priority tasks are able to sustain much better than the lower ones, conforming to our predictions. Figure 10-(b) depicts the number of helper nodes taken, which shows that lower priority tasks lose more helper nodes when resource is under intense competition.



(a)



(b)

Figure 10. (a) The performance of multiple ALM sessions and (b) The average number of additional nodes used.

6. RELATED WORK

Our work spans across a number of related fields: scalable monitoring services and its use in resource pool, optimizing ALM. We will discuss them in turn.

6.1 Resource Pool

To create a resourced pool, it is inevitable that a hierarchical structure to be adopted to ensure timely aggregation. Ganglia [31], for instance, is a two-level architecture in which IP-level multicasting is employed to gather statistics in one location, and the result is then aggregated to a central site. Gangelia is used in planet-lab. The GRIP and GRRP from the Grid toolbox [8] are protocols to allow interesting hierarchy to be built. Neither of them addresses the self-scaling aspect and requires manual configuration and administration. SOMO bears the most similarity to Astrolabe [28], a peer-to-peer management and data mining system. The difference between Astrolabe and SOMO was articulated in our earlier work [41]. Our contribution here is to point out the ingredients that make a wide-area resource pool feasible: the combination of the self-organizing capability of P2P DHT and an in-system, self-scaling monitoring infrastructure.

For interesting applications such as ALM, 1-dimensional metrics such as CPU power, memory and disk size etc. are still important, but for from complete. We show how metrics such as pair-wise latency estimation and up/downlink bandwidths can be derived in a completely distributed fashion by leveraging interactions inherent to P2P DHT. While this builds upon results of many previous works such as GNP [12]., Lighthouse [23] and PIC [3], to the best of our knowledge this is the first time that a comprehensive proposal is made.

We note that systems such as RanSub [19] establish an in-system aggregation and dissemination hierarchy as well. But a separate set of mechanism to generate the required statistics is still needed.

6.2 Optimize ALM using Resource Pool

Earlier work of ALM includes Aharoni's paper [1] and ESM [6]. Since then, quite a few other proposals and systems have emerged [2][33][34] including AMCast [34] from which our algorithm is derived. Researchers in P2P community quickly realized that application-level multicast maybe one of the showcases of P2P DHTs as well [5][26][44]. But neither of them is optimal; they either ignore and hence does not explore the potentials of a resource pool; or can not guarantee (for the time being) any QoS requirements. The DHT-based ALM proposals do not explore node heterogeneity. For all practical purposes, we believe that ALM with QoS requirements are restricted to small number of participants, and a centralized heuristic algorithms can guarantee QoS far better than trees that are built in an ad-hoc manner, and can leverage spare resources shall they be found. More importantly, given a resource pool, we have not only studied how to optimize one single ALM session, but also advocates a hands-off, market-driven approach to optimize multiple simultaneous sessions.

We emphasize here that ALM is only one of the applications for P2P resource pool. We argue that our two-step approach – application specific per task scheduling combined with market-driven fair competition coordinating among tasks is a far more distributed methodology than centralized matchmaking mechanisms.

7. CONCLUSION AND FUTURE WORK

To create a P2P resource pool, we need to combine the self-organizing capability of P2P DHT with a self-scaling, hierarchical in-system monitoring infrastructure. To achieve self-scaling and robustness, this infrastructure must be a logical hierarchy established in the virtual space created by DHT, and then mapped onto participants. In this paper, we illustrate that SOMO combined with DHT effectively creates a resource pool.

Our philosophy of utilizing the power of the resource pool is to take advantage of the on-time and accurate newscast via SOMO, install application-specific scheduler per each task and then take a hands-off, market-driven approach to coordinate among tasks with fair competition. We take the challenging application of ALM and demonstrate that significant benefits can be reached.

We are currently building a wide-area testbed to test the idea of P2P resource pool, and the ALM scheduling algorithm is one of the experiments we plan to run.

8. REFERENCES

[1] E. Aharoni and R. Cohen, "*Restricted dynamic Steiner trees for scalable multicast in datagram networks,*" *IEEE/ACM Trans. on Networking*, vol. 6, no. 3, pp. 286-297, Jun. 1998.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "*Scalable application layer multicast,*" *ACM SigComm 2002*, Pittsburgh, USA, Aug. 2002.

[3] M. Castro, M. Costa, P. Key and A. Rowstron. PIC: Practical Internet Coordinates for Distance Estimation. *Microsoft Research Technical Report.*

[4] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. *SOSP'03*, Lake Bolton, New York, October, 2003.

[5] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron A "*SCRIBE: A large-scale and decentralized application-level multicast infrastructure,*" *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1489-1499, Oct. 2002.

[6] Y. Chu, S. Rao, and H. Zhang, "*A case for end system multicast,*" *ACM SigMetrics 2000*, Santa Clara, CA, USA, Jun. 2000.

[7] The Condor Project, *http://www.cs.wisc.edu/condor*

[8] K. Czajkowski, S. Fitzgerald, I. Foster and C. Kesselman. Grid Information Services for Distributed Resource Sharing. *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, IEEE Press, August 2001.

[9] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris and I. Stoica. Wide-area cooperative storage with CFS. *19th ACM Symposium on Operating System Principles (SOSP)*, 2001.

[10] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. *Proceedings of HotOS VIII*, Schoss Elmau, Germany, May 2001.

[11] T. S. Eugene Ng, Y.-H. Chu, S. G. Rao, K. Sripanidkulchai and H. Zhang. Measurement-Based Optimization Techniques for Bandwidth-Demanding Peer-to-Peer Systems. In *Infocomm'03*, San Fancisco, CA, April 2003.

[12] T. S. Eugene Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *InfoComm'02*

[13] I. Foster and A. Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In *IPTPS'03*

[14] J. Frey, T. Tannenbaum, I. Foster, M. Livny and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*, San Francisco, California, August 7-9, 2001.

[15] The Globus Project, *http://www.globus.org*.

[16] J. Gray, "Distributed Computing Economics, " *Microsoft Technical Report*, MSR-TR-2003-24, Mar. 2003.

[17] KaZaA. http://www.kazaa.com/en/index.htm

[18] E. Korpela, D. Werthimer, D. Anderson, J. Cobb and M. Lebofsky. SETI@home: Massively Distributed Computing for SETI. *Computing in Science and Engineering*, 3(1), 2001.

[19] D. Kostić, A. Rodriguez, J. Albrecht, et al, "Using Random Subsets to Build Scalable Nethwork Services", *USITS'03*.

[20] J. Kubiatowicz, et al. OceanStore: An Architecture for Global-Scale Persistent Storage. *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000.

[21] K. Lai, Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. In *USITS'01*

[22] J. Ledlie, J. Shneidman, M. Seltzer, J. Huth. Scooped, again. In *IPTPS'03*

[23] M. Pias, J. Crowcroft, S. Wilbur, S. Bhatti and T. Harris. Lighthouse for Scalable Distributed Location. In *IPTPS'03*

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

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "*A scalable content-addressable network,*" *ACM SigComm 2001*, San Diego, CA, USA, Aug. 2001.

[26] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "*Application level multicast using content addressable networks,*" *3rd Interl. Workshop on Networked Group Comm.*, London, UK, Nov. 2001.

[27] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," *IEEE Infocom 2002*, New York, USA, Jun. 2002.

[28] R.v. Renesse, K. Birman, D. Dumitriu, and W. Vogels, "Scalable management and data mining using Astrolabe," *1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, USA, Mar. 2002.

[29] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, June 2003.

[30] A. Rowstron and P. Druschel, "*Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems,*" *18th FIFP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, Nov. 2001.

[31] F. D. Sacerdoti, M. J. Katz, M. L. Massie and D. E. Culler. Wide Area Cluster Monitoring with Ganglia. *Proceedings of the IEEE Cluster 2003 Conference*, Hong Kong

[32] S. Saroiu, K. Gummadi and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Conferencing and Networking* (San Jose, Jan. 2002)

[33] S. Shi and J.S. Turner, "*Routing in overlay multicast networks,*" *IEEE Infocom 2002*, New York, USA, Jun. 2002.

[34] S. Shi, J.S. Turner, and M. Waldvogel, "*Dimensioning server access bandwidth and multicast routing in overlay networks,*" *11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, New York, USA, Jun. 2001.

[35] T. Stading, P. Maniatis and M. Baker. Peer-to-Peer Caching Schemes to Address Flash Crowds. In *IPTPS'03*

[36] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "*Chord: a scalable peer-to-peer lookup service for Internet applications,*" *ACM SigComm 2001*, San Diego, CA, USA, Aug. 2001.

[37] Vahdat, A. and et al. *Self-Organizing Subsets: From Each According to His Abilities, to Each According to His Needs*. In IPTPS'02.

[38] E.W. Zegura, K.L. Calvert, and S. Bhattacharjee, "*How to model an Internet-work,*" *IEEE Infocom'96*, San Francisco, CA, USA, Apr. 1996.

[39] Z. Zhang, B. Lu, S. Shi and C. Yuan. Leafset Protocol in Structured P2P Systems and its Application in Peer Selection. *Microsoft Research Technical Report.*

[40] B. Zhang, S. Jamin, and L. Zhang, "*Host multicast: a framework for delivering multicast to end users,*" *IEEE Infocom 2002*, New York, USA, Jun. 2002.

[41] Z. Zhang, S. Shi, and J. Zhu, "*SOMO: Self-organized metadata overlay for resource management in P2P DHT,*" *2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, USA, Feb. 2003.

[42] B.Y. Zhao, J.D. Kubiatowicz, and A.D. Josep, "*Tapestry: an infrastructure for fault-tolerant wide-area location and routing,*" *U.C. Berkeley Technical Report*, UCB/CSD-01-1141, Apr. 2001.

[43] F. Zhou, L. Zhuang, B. Y. Zhao, L. Huang, A. Joseph and J. Kubiatowicz. Approximate Object Location and Spam Filtering on Peer-to-Peer Systems. *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, June 2003.

[44] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, and J.D. kubiatowicz, "*Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination,*" *11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, New York, USA, Jun. 2001.