

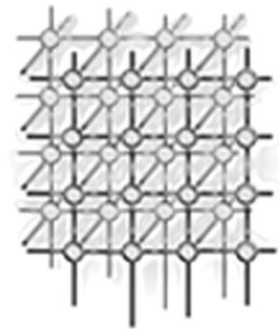
# How to measure a large open-source distributed system

Douglas Thain<sup>1,\*</sup>, Todd Tannenbaum<sup>2</sup> and Miron Livny<sup>2</sup>

<sup>1</sup>*Computer Science and Engineering Department, University of Notre Dame, 384 Fitzpatrick Hall, Notre Dame, IN 46556, U.S.A.*

<sup>2</sup>*Computer Sciences Department, University of Wisconsin–Madison, 1210 West Dayton Street, Madison, WI 53706, U.S.A.*

---



## SUMMARY

**How can we measure the impact of an open-source software package over time? When a system has no price, no purchase contracts and no buyers or sellers it can be difficult to judge its impact on the world. To explore this issue, we have instrumented the Condor distributed batch system in a variety of ways and observed its growth to over 50 000 CPUs at over 1000 sites over five years. Instrumentation methods include automatic updates by e-mail and user datagram protocol (UDP), annotated download records and a voluntary user survey. Each of these metrics has various strengths and weaknesses that we are able to compare and contrast. We also explore the ethical and legal issues surrounding automatic data collection. Surprisingly, we discover that objections to automatic data collection are higher among people that are *not* using the Condor software. We conclude with some practical advice for further research into the measurement of software systems. Copyright © 2006 John Wiley & Sons, Ltd.**

*Received 6 February 2005; Revised 14 September 2005; Accepted 24 September 2005*

KEY WORDS: measurement; open source; distributed systems; survey; mapping; Condor

## 1. INTRODUCTION

Large software systems resist measurement. As a system increases in size and number of human participants, it becomes increasingly difficult to locate all of the running components, to measure the desired properties or even to secure consent from all participants. System crashes, network failures and firewalls thwart the collection of data by automatic means. Noise can pollute data beyond all use.

Despite these problems, measurement is critical to the management of a large software project. Design changes to the system may hinge on key properties that must be measured. For example, the answer to ‘How many clients does each server typically support?’ can help a developer to make the right tradeoff between scalability and performance. The answer to ‘How many users still have version 6.0.3?’

---

\*Correspondence to: Douglas Thain, Computer Science and Engineering Department, University of Notre Dame, 384 Fitzpatrick Hall, Notre Dame, IN 46556, U.S.A.

†E-mail: dthain@cse.nd.edu



can help developers decide whether it is worthwhile to fix a bug or simply abandon a software version. The answer to ‘How many people actually use this software?’ can help to determine priorities when making financial decisions in a company or a funding agency.

Somehow, measurements must be made. In some settings, it is possible to make an accurate measurement when the participants are bound by a structural or a legal agreement. For example, a file server can easily measure all of its clients because they all must contact the file server in order to access data. A commercial software company might identify all of its customers by storing identifying information when software is purchased. In other settings—such as an open-source software project—such records of interaction may not be available.

Measuring software distribution is as much an exercise in sociology as in technology. Some participants in a system may be quite eager to share their presence and experience; we have spoken with a number of users that are quite eager to ensure that their systems are displayed on a world map. Others may have an incentive to ‘play’ the system. For example, SETI@Home [1] must deal with clients that fraudulently duplicate work to overstate their contribution to the system. Others may be more reticent to reveal their identity, location or characteristics. For example, Google [2] is known to employ many CPUs but the precise number is guarded as a valuable trade secret. Some participants may submit incomplete or bogus reports in an attempt at humor or anonymity. If every download report of Condor is to be believed, then our software is quite popular with several presidents of the United States.

In this paper, we explore how to measure the spread of one open-source software package deployed around the world. We draw from a number of data sources collected over a five-year period from the Condor distributed batch system. By comparing multiple data sources, we are able to identify the strengths, weaknesses and biases of each data source. The nature of the data prevents us from stating a precise measurement, such as the number of CPUs available at any given instant. However, it does allow us to produce a lower bound and describe properties of the system qualitatively.

We make three distinct contributions in this paper.

- (1) We offer some specific measurements of the Condor system that are of interest to its user community. For example, the median Condor pool size is only ten hosts. Such results do not generalize to other systems.
- (2) We make some general observations about Condor that may apply to other systems. For example, we propose a model of version lifetimes, observe a steady machine failure rate and infer user attitudes from a survey. It remains to be seen whether these observations generalize: other systems must be instrumented and monitored to determine this.
- (3) We describe the subtleties and tradeoffs of measuring software distribution over long time scales. For example, e-mail updates are biased towards large systems, while user datagram protocol (UDP) updates are biased towards small systems; downloads can be used to estimate running systems but only if they are annotated with unique user identities. Future work may apply these lessons in order to more effectively measure and understand large-scale software systems.

## 2. SUMMARY OF RESULTS

- On 2 February 2005, Condor was deployed on at least 60 428 CPUs in 1265 pools in 39 distinct countries.



- Condor has experienced two growth spurts: one driven by the addition of new pools and the other driven by new CPUs.
- The largest pool seen so far is 5608 CPUs, while for the past several years the median pool size has consistently been 10 CPUs.
- A software version takes 19 weeks to reach full deployment and has an average life time of two years.
- Voluntary download records are sufficient to estimate the size of the system.
- 77% of downloaders were willing to provide a valid e-mail address, determined by a successful delivery to that address.
- People that do *not* use the Condor software are much more likely to object to its data collection features, as measured by an online survey.
- Surveyed users overestimated the size of their own pools by 14%. This may reflect a high steady-state failure rate in computing clusters.
- By analysis of automatic e-mail updates, about 10% of machines are down at any given time, which is consistent with the previous point.
- Automatic UDP updates are biased towards small Windows systems, while e-mail updates are biased towards large Unix systems.
- Survey results suggest that about one half of operating pools are discovered by automatic updates. The remainder are unable or unwilling to report automatically.

### 3. OVERVIEW OF CONDOR

The Condor distributed batch system is designed to create a high-throughput computing system out of a collection of heterogeneous, autonomous, semi-available computing resources. The Condor software was first deployed at the University of Wisconsin in 1988. Condor continues to be used today for both research and production computing [3,4].

Figure 1 shows the structures of Condor relevant to this paper. A Condor *pool* consists of a *matchmaker* that oversees a number of processes representing resources and users. Each CPU in the system is represented by a *startd*. The *startd*'s job is to advertise the CPU to the matchmaker and to run batch jobs subject to the constraints of the CPU's owner. Each user in the system is represented by a *schedd*, which is responsible for queuing jobs and discovering *startds* willing to run them. In order to run a batch job, a user contacts a *schedd*, indicating the job to run and the required properties of a CPU. The matchmaker picks a compatible *schedd* and *startd* and introduces them to each other. The *schedd* then contacts the *startd* directly, transfers any necessary data and requests that the job be started.

A Condor pool is an administrative structure, not a physical constraint. The matchmaker serves as the guardian of the pool, controlling which agents and resources may participate and enforcing organizational policy, such as relative user priority. Within this structure, a Condor pool may have many different forms: it may be a multi-processor machine, a homogeneous cluster of commodity computers or a heterogeneous collection of workstations scattered across a wide area.

The set of all Condor pools in the world forms a loosely-coupled distributed system. In a procedure known as *direct flocking*, any *schedd* in the system can contact any matchmaker that it likes in order to discover CPUs and run jobs [4]. Of course, resource sharing is not automatic: access controls of

---

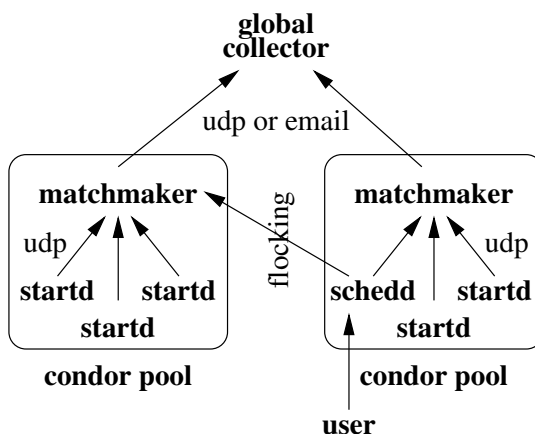


Figure 1. Structure of Condor. A Condor pool is composed of a matchmaker, startds and schedds. A startd represents a CPU available to run jobs. A schedd represents a user with jobs to run. The matchmaker introduces schedds to startds. Periodically, each matchmaker advertises itself to a global collector.

various kinds are enforced by matchmakers and startds alike. Administrators must explicitly admit remote users. That said, many people make use of flocking in order to access large numbers of CPUs. For example, in 2001, over 2500 CPUs were assembled via flocking in order to solve NUG-30, a long-standing optimization problem [5].

Each matchmaker periodically sends summary updates about the pool to a *global collector* at the University of Wisconsin. These updates are sent weekly by e-mail and once every 15 minutes via a single UDP packet. Each update describes the number of startds in the pool broken down by operating system and architecture and the software version of the matchmaker. The updates may be manually disabled by the user.

Due to the structure of Condor, each matchmaker knows the number and disposition of the participants in a system but not the work accomplished. Agents communicate directly with resources to execute jobs, so the matchmaker knows very little about the actual workload, except what it can infer from the resource requests issued by agents. Thus, this paper only explores the structure of Condor pools and not their workloads.

#### 4. OVERVIEW OF DATA

Our study is based on four measurement techniques: automatic updates by e-mail, automatic updates by UDP, records of software downloads and a survey of users. Each technique captures different aspects of the system and presents its own difficulties in interpretation.

Each running Condor system periodically sends update messages back to the collector at the University of Wisconsin. These are sent weekly by e-mail and every 15 minutes by UDP. Archives of all e-mail updates have been kept since January 2000. Although the global collector has always



been available for real-time query of running pools, we only began taking weekly snapshots of the collector in November 2003. Both e-mail and UDP updates can be lost or blocked in a variety of ways, for example, users may manually disable updates, e-mail services may not be configured at the sender, firewalls may block both types of updates and UDP packets may be dropped due to network congestion.

Downloads of the Condor software require each user to examine a posted license and then indicate consent to the terms by entering a name, organization and e-mail address. No effort is made to validate this input, users are free to enter whatever details they please. With each actual download of the software the system logs the self-reported identity of the downloader along with the version and platform of the package downloaded. This measurement does not give direct information about running systems but can be used to estimate the number of active users and in turn estimate running pools, as we will show below.

In December 2003, we conducted a survey of users in order to better understand privacy concerns as well as the biases of the other measures. A solicitation to participate in the survey was sent to the 2000 most recent downloaders of the Condor software that had been in possession of the software at least one week and 277 responses were received. The survey asked users about their experience with the software, their comfort with data collection and some details of their systems. It also yielded a number of Condor pools not detected by other means.

Using all of these data sources together, we may draw a comprehensive picture of Condor over a period of several years. Figures 2 and 3 are time-lapse maps of all known Condor pools from all available data sources. Each dot represents a single Condor pool and the area under the dot is proportional to the number of CPUs in the pool.

In order to plot each pool on a map, we need to convert network names into geographic coordinates. We use a two-step process similar to that described by Periakaruppan and Nemeth [6]. First, the WHOIS system is used to obtain a postal address corresponding to the domain name. Second, the postal address is converted into geographic coordinates via public databases. A United States postal database is used to resolve United States addresses to the nearest city, while addresses in other countries are resolved to the national capital. Pools with invalid names or WHOIS data are plotted in the south Pacific.

The ‘location’ of a pool must not be taken too literally. First, the various components of a single Condor pool may be spread across a wide area; we have simply plotted the matchmaker’s address. Second, the location used is not strictly that of the matchmaker itself, but of the postal address of the owner of the domain name. It is conceivable that such an address could be on the opposite side of the world from the pool itself.

Keeping these limitations in mind, several global trends may be observed. From 2001–2003, we may see that the largest Condor pools in the United States and Europe remained fairly static. In that same time period, there was a dramatic growth in the number of smaller pools, especially in the eastern United States, Japan and Europe. From 2003–2005, the mode of growth changed; large pools began to grow in size. Where there were previously only a small number of pools with hundreds of CPUs, there are now many pools measured in thousands of CPUs.

Figure 4 gives a more quantitative overview of the data. The solid line indicates the e-mail reports collected over a five-year period, steadily growing from a few tens to nearly 600 pools. The dotted line shows the *additional* pools discovered by the UDP reports, demonstrating that the e-mail technique alone certainly does not capture all known pools. The dot in January 2004 indicates the 81 additional

---

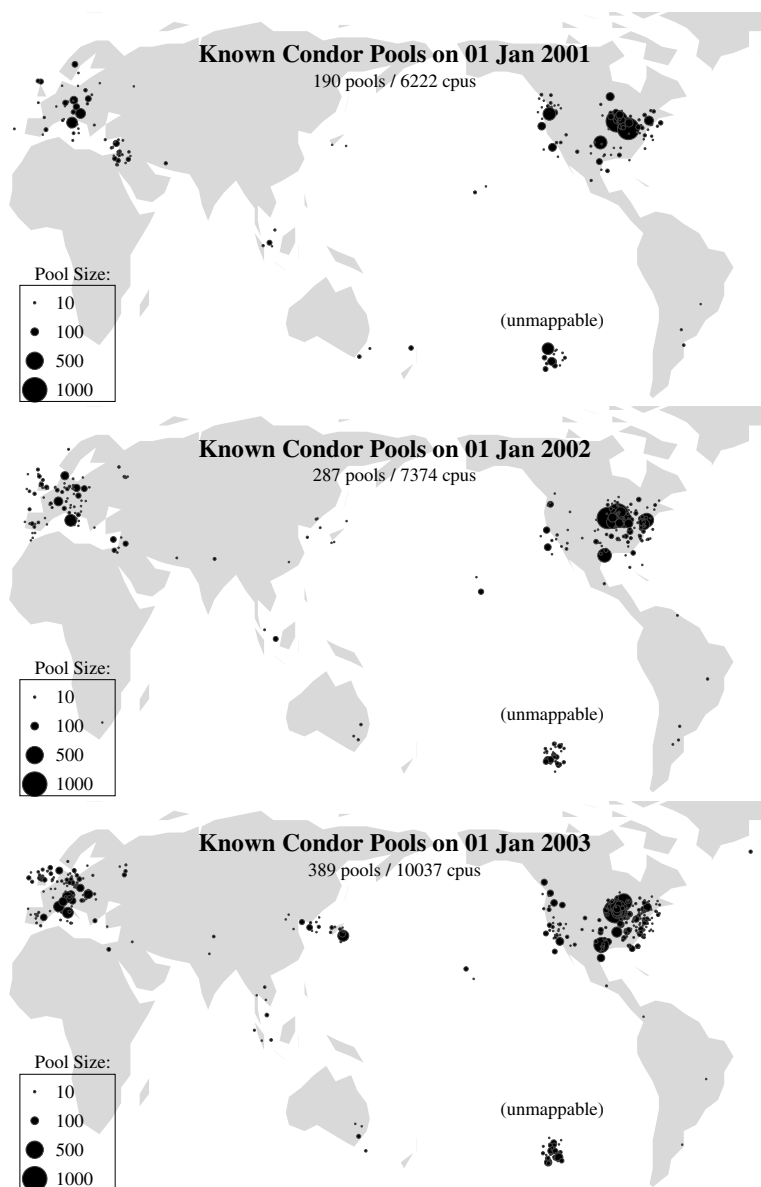


Figure 2. Time-lapse map 2001–2003. The distribution of Condor pools over time from all data sources. Each dot represents a single pool and the area of the dot is proportional to the number of CPUs in the pool. Within the United States, dots are plotted to the nearest city. Elsewhere, dots are plotted at the national capital. Each dot is scattered by a small random factor for visibility.

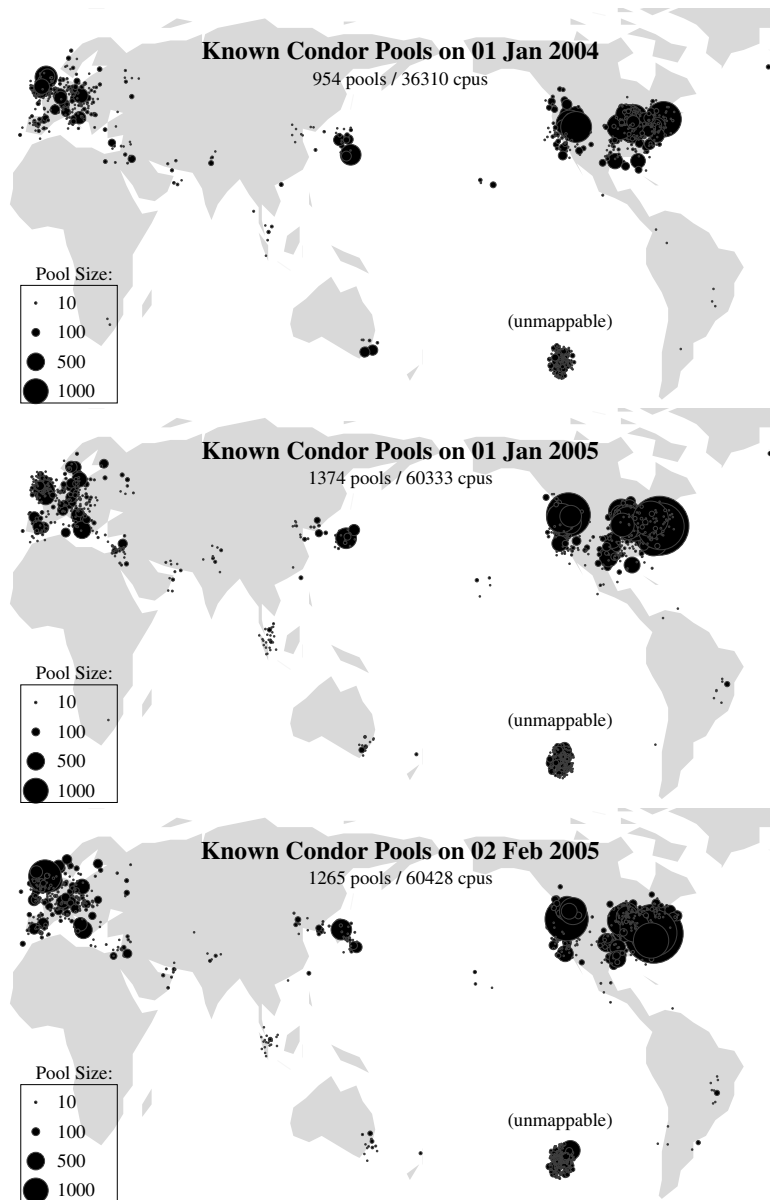


Figure 3. Time-lapse map 2004–2006. The distribution of Condor pools over time from all data sources. Each dot represents a single pool and the area of the dot is proportional to the number of CPUs in the pool. Within the United States, dots are plotted to the nearest city. Elsewhere, dots are plotted at the national capital. Each dot is scattered by a small random factor.

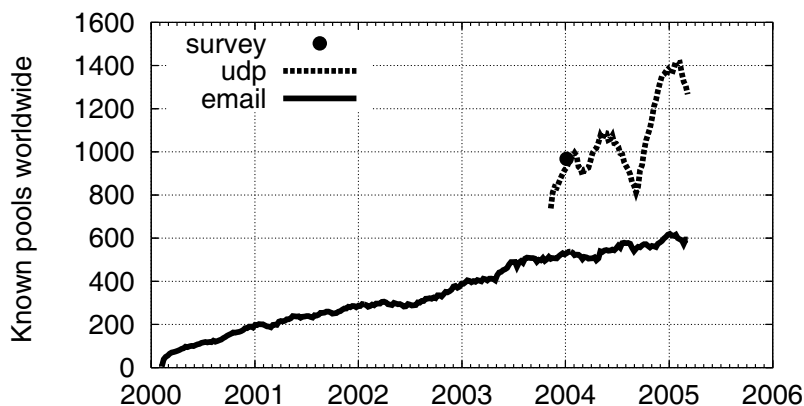


Figure 4. All known pools over time. The total number of Condor pools over time from all reporting methods. Records of e-mail updates have been kept from early 2000 through to 2005. Records of UDP updates which began in late 2003, capture many pools that e-mail does not. A survey in late 2003 captures a few more.

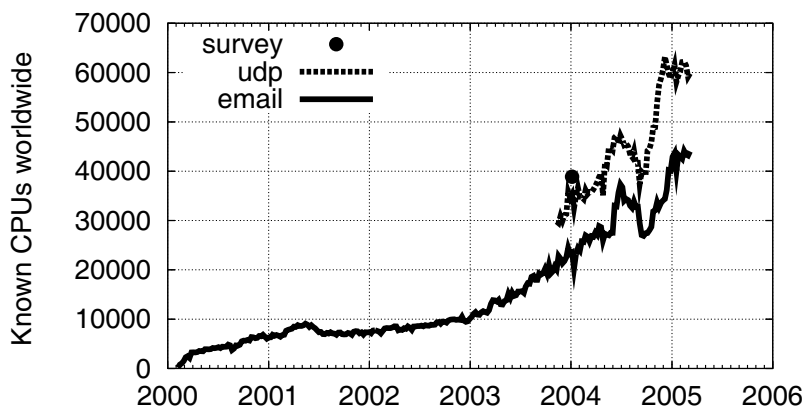


Figure 5. All known CPUs over time. The total number of Condor-managed CPUs from all reporting methods, much like Figure 4. Note that 2003 saw a doubling in the number of CPUs but a much smaller growth in the number of pools, indicating a general increase in pool size.

pools discovered by the user survey completed at that time. Figure 5 has a similar format, but shows the total number of CPUs.

Several trends may be observed by examining Figures 4 and 5 together. In 2001, the total number of pools increased by 50% but the total number of CPUs increased only by 18% and even suffered a setback in the summer. This is suggestive of restructuring of the system worldwide. One may imagine a fragmentation of administrative control without a corresponding growth in resources. Indeed, if you squint at the first two snapshots in Figure 2, you can see that the total amount of ink (CPUs) is roughly



constant while the dots disperse. Conversely, from January 2003 to January 2004, the number of CPUs tripled, while the number of pools increased by less than 50%. That is, existing pools expanded significantly but the creation of new pools slowed. It is tempting to draw analogies to biological cycles (e.g. mitosis) but we will have to wait another five years to see if the cycle repeats.

## 5. ETHICAL AND LEGAL MATTERS

The ethical concerns of computing have come into the public eye as the commoditization of computing, the spread of networking and the aggregation of data have all taken root in public life. It would be easy but disingenuous to argue that our benevolent aims disqualify us from these concerns. Before embarking on a detailed discussion of our data, we will consider some of the ethical and legal matters of data collection and publication.

We are obliged to respect the privacy that users expect in the management of their own affairs. Revealing the nature and details of a machine may potentially expose the owner to harm from various sources. The precise number of CPUs employed by a company might be a valuable piece of corporate espionage. The network address of a critical service might be just enough information to enable all manner of cybercrime. Of course, users should be able to choose their degree of exposure, balancing their needs appropriately. To this end, we have made a significant effort to publicize our actions and intent as well as to allow users to opt-out if desired.

Several passive sources of information are available to users. The nature and content of the data collection messages—and how to disable them—are detailed in the user manual and in the configuration file for the software. In addition, a privacy policy stating how we will (and will not) use such data is stated on the main Condor Web page, the download Web page and in the printed manual. Various maps and charts based on current data are available on the Web accompanied by a description of the data collection and how to disable it.

We have made active attempts to reach users as follows: in the survey described below, a description of the data collection was included, along with a link to the relevant section in the manual and an example of a map created from the data. At our annual users meeting in March 2004, we made announcement of these data collection activities, presented a handout with preliminary results and solicited users for feedback. It is worth noting that of those in attendance, one objected to the procedure and was happy to learn that it could be disabled. Five users came forward in order to make sure that their pools would appear on the map.

Two users discovered our data collection techniques independently with tracing tools and posted their discoveries on the public Condor mailing list in 2004. Members of the Condor staff responded with explanations and pointers to the published material. Although both posters were quite unhappy to have discovered this behavior through instrumentation, neither interaction drew any response from other readers.

Our use of the WHOIS system for locating network addresses may be controversial. In the early days of the Internet, a strong public binding between the virtual and physical worlds was taken for granted: a single server WHOIS at the Stanford Research Institute (SRI) contained a global database of all people, networks and institutions associated with the network [7]. With the expansion of the Internet, many of these roles could not be continued but the role of binding domain names to their

---



real-world owners continued. The single WHOIS server has been fragmented into multiple servers, one for each top-level registrar.

A variety of people and organizations have expressed opinions about the proper use of the WHOIS database. Some are in favor of expanded use of WHOIS in order to improve the trustworthiness of transactions on the Internet. For example, the United States Federal Trade Commission has lobbied Congress in favor of requiring accurate WHOIS data for the sake of law enforcement and consumer protection [8]. The Internet Corporation for Assigned Names and Numbers (ICANN) also recommends exposure of WHOIS data but primarily to administrators in order to ensure the stability of the network [9].

On the other hand, several parties have expressed concern that public access to WHOIS invites identity theft and attack by spammers, stalkers and rogue government agents. This position is held by the Association for Computing Machinery (ACM), which joined a public letter to ICANN, requesting restrictions on WHOIS searches for reasons of privacy, while agreeing that WHOIS is a necessary tool for the resolution of network problems [10]. The Electronic Frontier Foundation (EFF) holds a similar position and suggests that rights be balanced by mutually revealing the contact information of domain holders and WHOIS users to each other [11].

Domain name registrars have a third interest. As current registrars are obliged to provide access to WHOIS data online, they must also absorb the operational expense of the databases. The simple WHOIS protocol is neither as scalable nor as efficient as the domain name service itself. Consequently, registrars have confronted heavy WHOIS users with service denial and even legal challenges when queries begin to impose a noticeable load. WHOIS servers now ‘tarpit’ heavy users by rejecting their connections for a limited time.

We have attempted to respect each of these competing concerns in our use of the WHOIS system. Even though WHOIS data are already public, we have taken care not to expose the data in collected form without anonymization. In constructing maps, we have added random noise to locations so that precise locations (and, hence, identities) cannot be inferred. We also throttled queries of WHOIS servers so as not to produce excessive query loads.

A number of commercial concerns provide databases and online lookup services that map Internet addresses to geographic coordinates using proprietary techniques. Although such services could be used to generate our maps, they essentially move the privacy concern from the public to the private sector. Furthermore, we would have little or no knowledge of the source or accuracy of the data. By using the public WHOIS database, whatever its imperfections, we constrain ourselves to known data used within the constraints of public policy.

We suspect that a legislative oversight of data collection in distributed systems is quite likely in the near future. For example, a recent bill [12] passed by the United States House of Representatives intended to combat ‘spyware’ would outlaw automatic collection of data for the purpose of displaying advertisements or for discovering information that personally identifies a user. (Condor does not currently fall under the scope of this bill in either category.)

## 6. DATA ANALYSIS

So far, we have given an overview of Condor using all data together. In this section, we will draw out finer details from each data source independently.

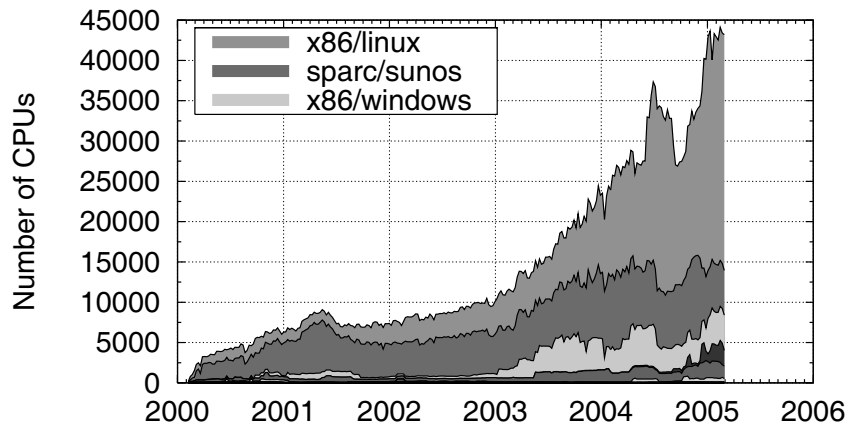


Figure 6. Deployment of platforms. The deployment of various platforms over time. Each shade represents a distinct cpu/os combination. The three largest platforms, starting from the top, are x86/Linux, parc/Solaris and x86/Windows.

### 6.1. E-mail updates

We consider a large number of time-dependent aspects using only the e-mail data because this rich data source has been collected consistently for the longest amount of time. The UDP mechanism effectively collects the same information as the e-mail mechanism. Due to the short time for which we have collected UDP data, we will not present an analysis of the UDP data separately. As we will see below, the UDP mechanism collects data from a substantially *different* set of pools.

Figure 6 shows the total number of CPUs discovered by the e-mail collection process over time, broken down by platform. Each shade indicates a distinct hardware/software combination. The three major platforms, from the top, are Linux, Solaris and Windows. A number of other minor platforms are barely visible underneath. Note that the total number of CPUs showed steady growth from 2000 to 2003, where a significant growth spurt began. This growth spurt was carried initially by several thousand Windows machines added when the first production version for Windows was released. Later growth in 2004 was carried by an increasing number of Linux machines as a result of the large pool expansion phenomenon discussed above.

Several large fluctuations in the number of CPUs reported by e-mail occurred over the summer of 2004. This was the result of several events in close succession. In June, a number of large Linux-based pools at academic institutions were brought online. In July, about 500 Windows CPUs went offline for reasons unknown. In August, e-mail reports stopped arriving from several large Linux pools. Checking with the owners of these pools, we discovered that they were *transmitting* correctly but the e-mails were simply not arriving. In October, the e-mails suddenly began arriving again.

At this point, we have not discovered the precise reason for the loss. One reasonable hypothesis is that ISP-level mail filtering or other restrictions were preventing mail delivery. Although the local operations at both sides are not filtering these reports, many upstream providers have mechanisms for

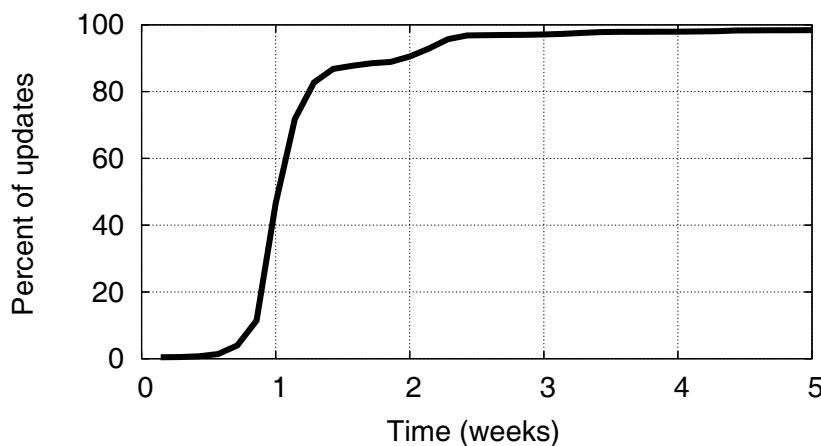


Figure 7. Update arrival distribution. The cumulative distribution of interarrival times for e-mail updates. Updates should be transmitted weekly but are dispersed due to the vagaries of e-mail. Note that about 10% of updates are simply lost.

reducing unwanted mail traffic. Condor e-mails might be identified as spam or the set of machines authorized to send outgoing mail may have been temporarily restricted. However, we are unable to verify these hypotheses. Some amount of noise in the measurement is inevitable at this scale.

More widespread evidence of e-mail loss can be found by examining e-mail interarrival times, shown in Figure 7. One interarrival is the time elapsed between e-mails received *from a given pool*. The figure shows the cumulative interarrivals from all pools over the entire study. As might be expected, the majority of interarrivals (about 90%) are centered around one week. There is a variation of several days on either side due to clock skew and the delays inherent to e-mail delivery. Of the remaining 10%, about 90% arrive in the next week. A similar recursive pattern repeats in remaining weeks.

A surprising observation can be made from this graph: *a random 10% of weekly e-mail updates are lost*. The implications of this observation are subtle and require some explanation. We claim that the e-mails are indeed *lost* because of the shape of the graph. If the e-mails were simply delayed, the interarrival graph would result in a smooth asymptotic curve up to 100%. Rather, 10% simply never arrive; the pool is not heard from until it attempts to transmit the next week's message, resulting in the 90% 'bumps' on the graph. We claim that the losses are *random* because we do eventually hear from those pools whose updates are lost. If the losses were consistent (i.e. mail was deliberately blocked or filtered as spam), then we would simply never hear from those pools at all.

Our best hypothesis for this observation is that 10% of matchmakers are down at any given time, which is a fairly grim hypothesis for any software system. In this context, *down* must have the fairly broad meaning of *unable to queue mail for delivery*. In order to test this hypothesis, we must add information about recent failures to the weekly update. This would allow us to distinguish between failures of the software, the network, the machine and perhaps the interception of mail. Fortunately, the random loss of a small percentage of updates does not have a dramatic effect on the analysis techniques that we use.

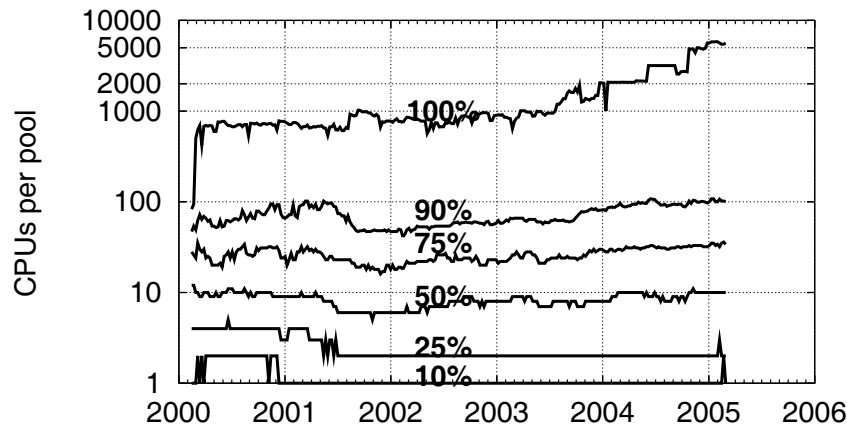


Figure 8. Distribution of pool sizes. The distribution of Condor pool sizes over time. Each line follows a percentile of pools ordered by size. Note that the largest Condor pools have grown from several hundred to several thousand CPUs, while the median Condor pool size has remained at 10 or fewer CPUs.

Figure 8 shows the distribution of Condor pool sizes over time. Each line tracks a percentile of the distribution. For example, the 100th percentile—the largest pools—has grown from several hundred to several thousand CPUs. The 50th percentile—the median—has hovered at or below 10 CPUs. That is, half of all Condor pools are quite small. Surprisingly, the 10th percentile is exactly one machine. This number is hard to interpret because it may include two configurations. Casual users are likely to install Condor on one CPU in order to evaluate the software. However, sophisticated users also set up ‘personal’ Condor pools that simply flock with larger pools. Condor has traditionally focused on throughput to the detriment of response time; it is designed to maximize the productivity of large pools. However, this graph suggests that most users employ Condor in relatively small configurations. Some greater attention to performance in small pools is warranted.

Figure 9 shows the total number of pools discovered by e-mail, broken down by the software version of the matchmaker. (The e-mail updates do not reveal the software version of all components in a pool.) Each shade in the graph indicates one version of the software. As there are so many versions, it is not practical to label them all. A number of facts may be observed by simple visual inspection. No single version has ever dominated the worldwide system. In fact, no single version has ever accounted for even half of all machines, except for a short period in 2002. Versions, even very old ones, persist for years at a time. At any given time, tens of different versions are in use at once. Thus, backwards compatibility must be a serious concern for this type of software. If new software is incompatible with old, problems will persist for years between many users.

We would like to draw more general conclusions about the nature and lifetime of software versions in the wild. Of course, most of our data is specific to Condor and is dependent on how often versions are released and how users react to announcements of bugs and features. However, we believe that some general conclusions can be drawn by putting the data into more abstract form.

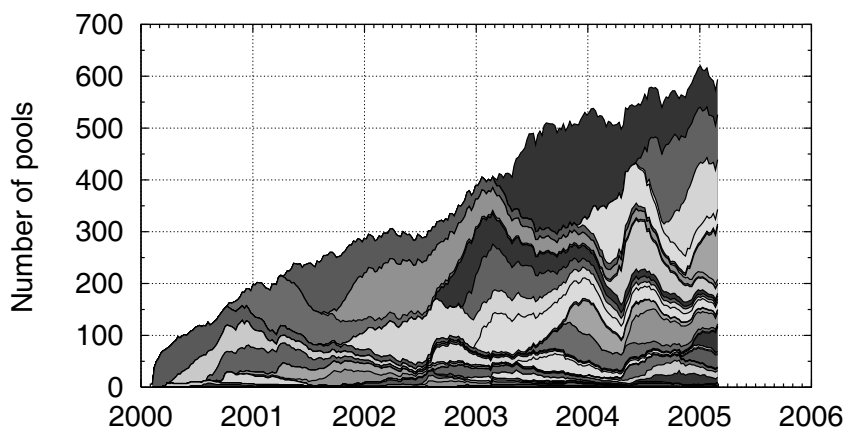


Figure 9. Deployment of versions. The deployment of software versions over time. Each shade represents the number of pools running a distinct version. Each version has a very long lifetime in the wild, so there are a large number of deployed versions at any given time.

We begin by removing any software versions that were not widely deployed as these have unusual patterns. We are left with 13 software versions that ran at 50 or more sites. Each version saw a different total deployment, depending on its popularity and the overall size of Condor at the time. So, we normalize each version against its maximum deployment and shift each version in time so that time zero corresponds to the first download of that version.

The result is Figure 10, which shows the deployment curve of all significant versions in time relative to the first download. Although there is some variation, there is a common shape to each version. In order to compare versions, we define several life phases of each version. In the *growth* phase, the version rapidly expands as pools are created with (or converted to) the new version. In the relatively short *peak* phase, the version hovers at or near its maximum deployment. In the *decay* phase, the version shrinks as pools upgrade or are decommissioned.

Table I gives specific values for the phases of each version shown in Figure 10. The *growth* phase is measured (in weeks) from time zero to 90% of the maximum deployment. The *rinterval* column shows the release interval, defined as the number of weeks between the release of named version and the next release. The *peak* phase is measured from the first to last instances of 90% deployment. The *decay* phase is given by *decay-50* which gives the time from the end of *peak* to 50% of maximum, while *decay-10* gives the time from 50 to 10%. The term *life* measures from time zero to a decay of 10%, while *max* gives the maximum number of pools deploying that version.

We were surprised to find such a long growth phase. An average of 19 weeks to grow to peak deployment has significant consequences for bug fixes. A routine bug, fixed and released in an ordinary version, will take about five months to reach maximum dispersion. It was also surprising that the peak phase is relatively small, only 10 weeks on average. Every version spent the majority of its time growing or declining and very little time in a stable state. The average lifetime of a version is about two years and some lived over three.

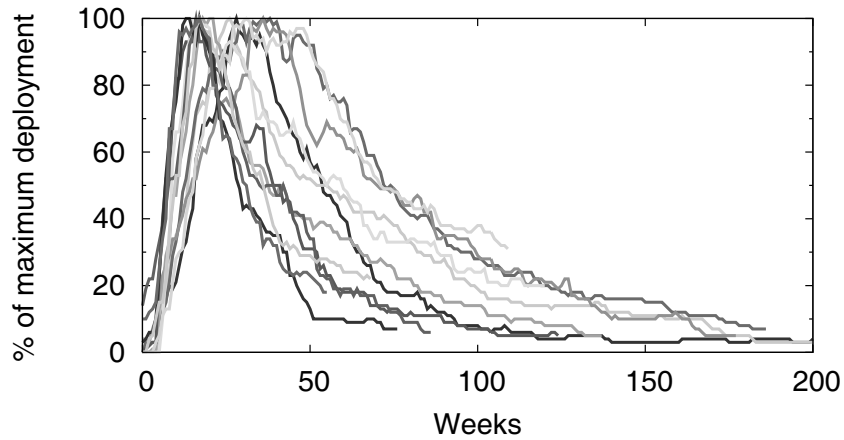


Figure 10. Version lifetimes compared. Each line shows the growth and decline of one software version over time. Because each version reached a different size audience, each is normalized to 100 percent of its maximum deployment.

Table I. Empirical parameters of 13 widely deployed software versions. Each column indicates the number of weeks spent in each stage of life. The *max* column gives the maximum number of pools running the version at one time.

Version	<i>growth</i> (weeks)	<i>rinterval</i> (weeks)	<i>peak</i> (weeks)	<i>decay-50</i> (weeks)	<i>decay-10</i> (weeks)	<i>life</i> (weeks)	<i>max</i> (pools)
6.1.12	17	15	22	9	71	119	53
6.1.16	26	22	11	16	40	93	100
6.2.0	22	6	7	26	112	167	114
6.2.1	24	8	27	23	99	173	106
6.3.1	32	22	12	26	93	163	89
6.4.1	17	12	6	19	74	116	96
6.4.3	12	2	6	23	48	89	94
6.4.5	14	3	14	31	61	120	75
6.4.7	27	4	23	21	38	109	207
6.5.3	14	2	6	22	35	77	63
6.5.5	12	7	8	8	41	69	65
6.6.0	16	12	6	14	32	68	118
6.6.1	11	4	9	11	24	55	97
Average	19	9	12	19	59	109	98
	±6	±6	±7	±7	±28	±39	±38

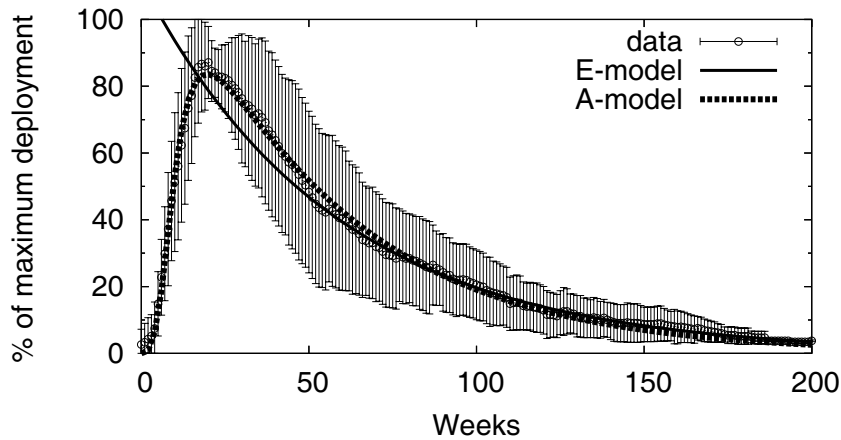


Figure 11. Version lifetime model. The combined 13 lifetime patterns from Figure 10. At each point, a circle shows the average of all versions and a bar shows the standard deviation. The E-model fits only the decay process, while the A-model fits the entire lifetime.

Intuitively, one might expect that the growth phase would be limited by the release of the next software version in sequence. In practice, version succession is not quite so simple. Like many other software packages, Condor is developed and released along two parallel tracks, a stable track (indicated by an even second number) and a development track (indicated by an odd second number). The stable track generally only updates the software for bug fixes, while the development track adds the latest features and ports. Releases along each track are often interleaved. Users respond to these two tracks differently. Some stick with one track, while some upgrade to the latest software, regardless of the track. In addition, releases on each track representing small changes may occur in rapid succession. We expect that few users install every software version, preferring to upgrade only when a desired feature becomes available. As multiple versions are simultaneously available, users may continue to install and use trusted older versions, even as newer versions become available. As a result, there is no direct relationship between release interval and the end of a version's growth. This can be seen by comparing the *growth* and *rinterval* phases. In all cases, the growth phase exceeds the next release interval, sometimes by only a few weeks but often by a large margin.

In order to create an analytic model of version lifetimes, we combine all of the version data together and produce an average and a percentage error of deployment at each time since download. This is shown in Figure 11. Two models are fitted to these data: an exponential model (E-model) and an asymptotic model (A-model).

It is reasonable to assume that the decay process is exponential. At a certain point past the growth phase, there are no new additions to the population and at each time step a certain fraction of the population decides to decay by upgrading. Thus, the E-model is simply

$$E(t) = a e^{-t/c} \quad (1)$$



Fitting the E-model to the data starting at 50 weeks, yields the following parameters:

$$\begin{aligned}a &= 144.85 \pm 3.04 \\c &= 48.85 \pm 0.64 \\ \chi_v^2 &= 2.87\end{aligned}$$

$c = 48.85$  corresponds to a half-life of 35.4 weeks. That is, once a version begins its slow decay, the number of pools with that version will halve every 35.4 weeks.

We would also like to have a model for the entire process including growth and decay. A number of standard distributions with this general shape do *not* fit the data well. For example, Poisson and Weibull distributions have a much shorter tail and are constrained by normalization factors that are irrelevant to this domain. Neither can fit both growth and decay.

However, we can argue informally for a model that empirically matches the data. A version has a total number of deployments that it eventually reaches. Initially, the growth is fast as new users discover the new version but eventually slows and approaches a maximum value as the set of new users is exhausted. An expression that satisfies these criteria is  $a(1 - [1/(t/b)^n + 1])$  where  $a$  sets the maximum value,  $n$  controls the shape of the curve and  $b$  is a horizontal scale factor. As the growth function approaches a constant value, we may simply multiply by the exponential decay factor already shown. Thus, we propose  $A(t)$  as a model of version deployment:

$$A(t) = a \left( 1 - \frac{1}{(t/b)^n + 1} \right) e^{-t/c} \quad (2)$$

Figure 11 shows that the A-model fits the data well with the following parameters:

$$\begin{aligned}a &= 151.52 \pm 2.29 \\b &= 11.52 \pm 0.16 \\c &= 47.85 \pm 0.52 \\n &= 2.82 \pm 0.08 \\ \chi_v^2 &= 3.62\end{aligned}$$

We may also consider the lifetime of Condor pools themselves. We define the lifetime of a pool to be the time between the first and the last updates received. Figure 12 shows the distribution of pool lifetimes obtained in this manner. The results of this figure were somewhat of a surprise: we expected a more dramatically bimodal distribution. About 20% of pools are heard from once and then never again. This is likely due to users that try the software once and then discard it. Most pools have a lifetime ranging from weeks to months. Few live longer than one year. About 25% of pools were still living today, so their lifetime is unknown.

## 6.2. Downloads

For many software projects, it may not be possible to measure running systems directly. Perhaps the software is already deployed without instrumentation, the communication medium is not suitable or the user community is simply not comfortable with the idea of automatic updates. However, nearly all projects can make a record of software downloads. Given only the latter, is it possible to make an estimate of running systems? We will demonstrate that this is possible.

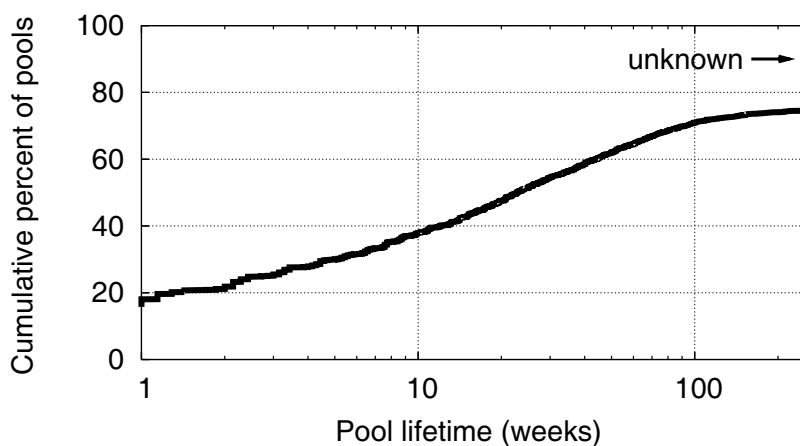


Figure 12. Pool lifetime distribution. For example, about 35% of pools live 10 weeks or less, while 75% of pools live 100 weeks or less. Note that about 25% have an unknown lifetime, i.e. they are still alive today.

That said, downloads are a potentially dangerous method of evaluation. On the one hand, they are easily measured; almost any sort of server can produce an access log. On the other hand, the raw number of downloads can be wildly skewed from the metric of interest. One active user might download the software once in 10 years, while the Slashdot effect might produce thousands of downloads by disinterested people.

Figure 13 demonstrates the noise present in this metric. Each point represents one unique user that has downloaded the software more than once. (Recall that user identity is requested before downloading.) The  $x$ -axis indicates the average interval between downloads, while the  $y$ -axis indicates the total number of downloads. Although this graph clearly captures a wide array of downloading behavior the results can be grouped roughly into the two visible clumps. The left clump represents users that are repeating downloads rapidly, perhaps retrying failures or obtaining all available versions at once. The right clump represents users that are more regular visitors. Those low and to the right visit every few months or years, while those high and left may be automatically downloading the software on a weekly or monthly basis.

This figure has some surprising outliers. For example, one user downloaded the software several hundred times at 10-s intervals. In this case, the user had established an automatic download of the software and left it running for some time before realizing the interval was misconfigured. A number of users are noted as downloading the software several hundred times at one-day intervals. These represent mirror sites that keep their copies updated at one-day intervals.

The scatter graph is effective at showing the noise and outliers in the data. Two cumulative graphs do a better job of drawing more general conclusions. Figure 14 shows the total number of downloads per user. Note that, of the 15 000 uniquely identified downloading users, about 6000 visited only once, while about 8000 visited a handful of times and about 1000 visited 10 or more times.

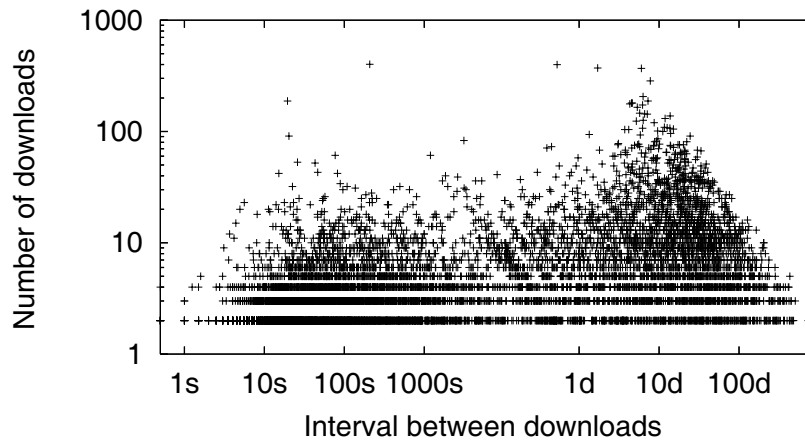


Figure 13. Frequency and volume of downloads. The distribution of frequency and volume of downloads by user. Each point represents one unique user. The  $x$ -axis indicates the average time between downloads, while the  $y$ -axis indicates the total number of downloads.

Figure 15 shows the average interval between downloads for the users that visited more than once. Three distinct domains are visible, for which we may suggest some plausible behaviors. About 4500 users returned over the course of seconds or minutes, perhaps to download multiple versions or platforms. About 1000 returned over hours or days, perhaps to try a different version after attempting one installation. About 3000 returned over the course of weeks or months, perhaps to upgrade a running pool.

It is not possible to leap directly from downloads to running systems as we would like to do. One recorded download might result in many installations, if the download is replicated to multiple sites. On the other hand, many downloads by a single user might not result in any installations. However, if we assume a *typical* relationship between downloads and deployment, we may obtain a reasonable estimate of deployments in the aggregate. Furthermore, we are able to show that this estimate corresponds closely to measured deployments.

Our aggregate estimate works as follows. We assume most Condor pools are overseen by one person who is responsible for downloading and installing software. Furthermore, repeated downloads over a long period of time are an indication of continuing interest and active use. The number of downloads is not as critical as the time span; 10 downloads spread over a year mean much more than a burst of 10 downloads in one day. This reduces the effect of outliers that download repeatedly.

We define two simple estimates based on download activity, both demonstrated in Figure 16. The *low estimate* assumes that a pool is alive from the time of a user's first download to the last. Thus, a download in January 2003 followed by a download in June 2003 by the same user indicates one pool alive in that interval. The *high estimate* assumes that pool is alive from the time of a user's first download to the last plus an additional download, using the average download time as the interval. Thus, a download in January 2003 followed by a download in June 2003 by the same user indicates one pool alive from January 2003 to January 2004. The assumption of the high estimate is that the past

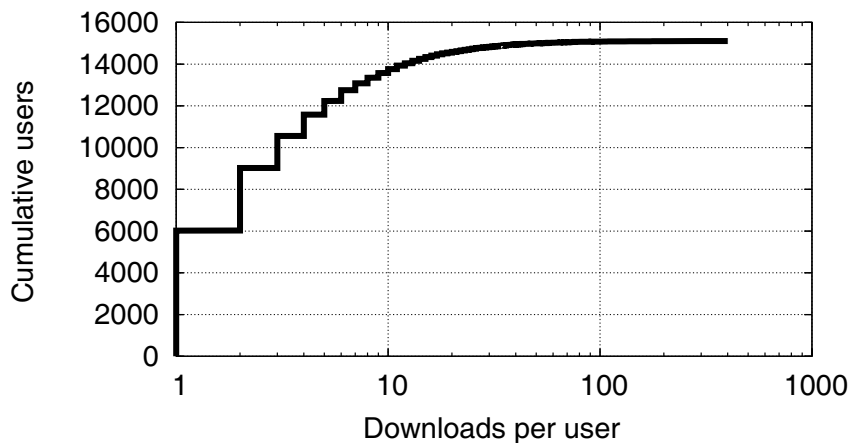


Figure 14. Distribution of downloads by user. The number of downloads made by each observed user. About 6000 users downloaded only once and another 8000 less than 10 times. About 1000 users downloaded 10 or more times.

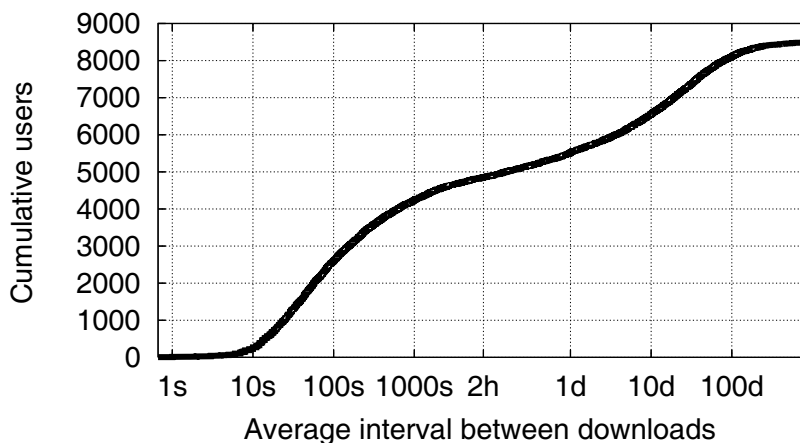


Figure 15. Distribution of download intervals by user. The average interval between downloads for each user that visited more than once. About 4500 users returned in seconds or minutes, about 1000 returned in several hours, and about 3000 over days, weeks or months.

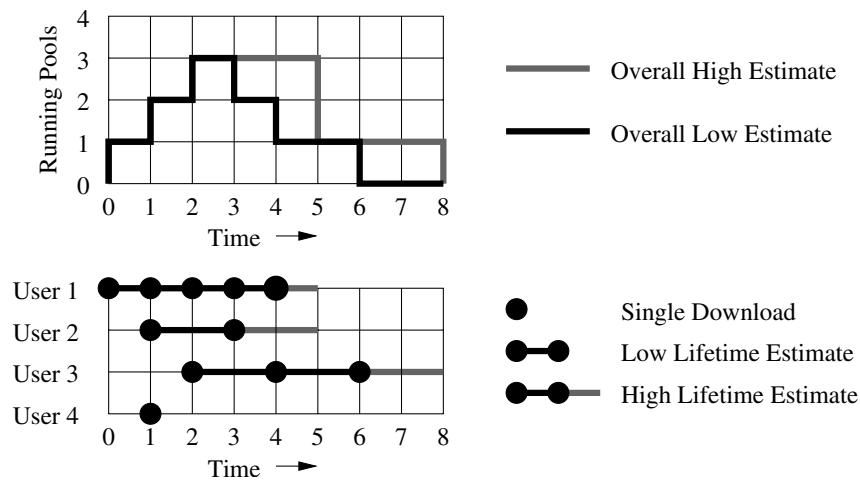


Figure 16. Simple estimation method. To estimate the number of active pools, we assume that each active downloader represents one active pool. The low estimate assumes a pool is alive between the first and last download of a user. The high estimate assumes a pool is alive for the low estimate plus one more download.

is like the future; a regular customer will return. Note that for both estimates a single download by one user has no effect.

Figure 17 shows how the download-based estimates compare with actual measurements of Condor pools. As can be seen, both estimates are in the right order of magnitude. The low estimate exceeds the number of pools actually discovered by e-mail from 2001–2004 but by a smaller fraction than the actual UDP measurement, performed in 2004. The low estimate drops off markedly in 2004 because of the way that active pools are computed. If the person representing that pool only makes a download visit every six months then that pool loses its representation in the last six months. The high estimate shows this effect as well but to a lower degree. As data continues to be collected the falling curve moves to the right. In the period 2004–2005, the high estimate is a good approximation of the combination of all data sources. We cannot verify whether the high estimate is valid in earlier years but it is plausible to assume that a similar fraction of pools were not measured by the e-mail technique for those users.

Thus, we have shown that it is possible to give a reasonable estimate of a distributed system using only download records annotated with user identities. The total downloads by themselves do not yield a meaningful number but voluntary user identity allows us to reduce duplicates and identify consistent customers.

### 6.3. User survey

To complement the automatic collection activities we also conducted an online survey. On 1 December 2003, we sent a survey solicitation by e-mail to the most recent 2000 users that downloaded the Condor software at least one week previously, asking each to fill out an eight question survey on our Web site. Of these, 468 (23%) were returned as undeliverable, indicating that 77% of downloaders at least

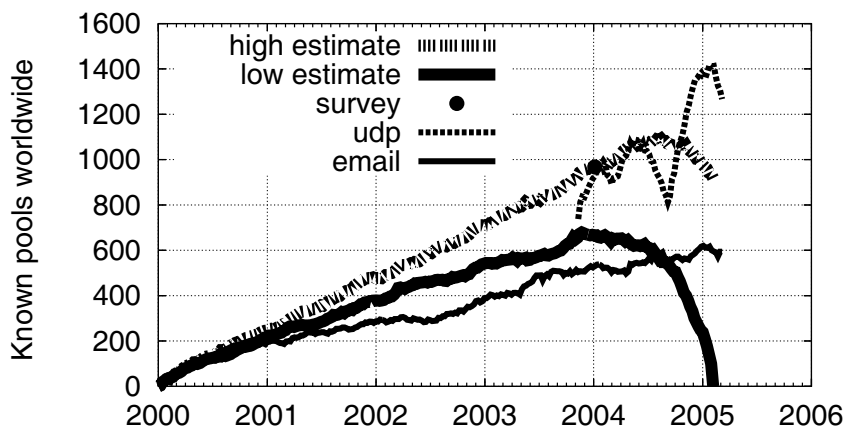


Figure 17. Estimates from downloads. An estimate of running pools based on downloads. The high and low estimates are computed using the method shown in Figure 16. The high estimate corresponds closely to the pools measured using all methods.

provided a valid and active (although not necessarily correct) e-mail address. Of those valid addresses, 277 (18%) filled out the survey. This gives a margin of error of 5.9% with 95% confidence.

It is well known that self-selected surveys tend to attract people with strong opinions. As an example of this, many users typed long paragraphs into a very small text box for general comments at the end of the survey. Two short examples are: ‘My entire experience with your software . . . was very infuriating!’ and ‘I think Condor is the greatest thing since sliced bread!’

Three survey questions are of particular interest to this paper. One question was intended to evaluate the relationship between downloads and other reports of Condor activity.

### 3—How far did you actually get?

- |                       |   |            |            |
|-----------------------|---|------------|------------|
| <input type="radio"/> | I did not download Condor                     | <b>3</b>   | <i>1%</i>  |
| <input type="radio"/> | I downloaded Condor                           | <b>25</b>  | <i>9%</i>  |
| <input type="radio"/> | I installed Condor                            | <b>23</b>  | <i>8%</i>  |
| <input type="radio"/> | I tested Condor                               | <b>77</b>  | <i>27%</i> |
| <input type="radio"/> | I used Condor to accomplish<br>some real work | <b>135</b> | <i>48%</i> |

(Bold numbers indicate the absolute number of responses, while italics give the percentage of total respondents. The numbers do not add to 277 because not all users answered all questions.)

This question was included with the intent of determining how many downloads translated into actual running pools. However, given the response rate and the knowledge that the survey attracted users with extreme views, we can draw few conclusions from the question directly. However, it *was* useful for calibrating responses to the following question.



Before publishing this paper, we wanted to gauge how sensitive users might be to our data collection and publication efforts. After explaining how Condor collected data and giving an example of a map like those in this paper, we asked the following question.

**6—If the University of Wisconsin published a map of Condor pools worldwide, similar to the one below, would you want your Condor pool shown on the map?**

- Yes **143** 51%
- No **33** 11%
- Not sure **41** 14%
- Don't have a Condor pool **44** 15%

We included the final 'Don't have' option assuming that those without Condor pools would not express an opinion. Although this result shows about one-half of those polled were willing to be shown on a map, 25% still answered 'No' or 'Not sure'. Given only these data, we felt that it would be unacceptable to publish a map without significant efforts to satisfy those that would not want to be included. However, by correlating questions 3 and 6, we learned something more about our user population. The following table shows how respondents answered both questions.

	<b>Answer to Question 6</b>			
	<i>Don't have</i>	<i>No</i>	<i>sure</i>	<i>Yes</i>
<b>Answer to Q. 3</b>				
<i>Nothing</i>	2	5	0	0
<i>Downloaded</i>	10	3	3	7
<i>Installed</i>	6	14	5	9
<i>Tested</i>	18	11	15	30
<i>Used</i>	8	0	18	95

If we lump together the first four rows as 138 users that did *not* use Condor for real work, we may compare them to the 121 users that did.

	<b>Answer to Question 6</b>			
	<i>Don't have</i>	<i>No</i>	<i>sure</i>	<i>Yes</i>
<b>Answer to Q. 3</b>				
<i>Did not use</i>	36	33	23	46
<i>Used</i>	8	0	18	95

A striking observation can be made from this table. Of the 138 people who did *not* use Condor for real work, 33 (24%) answered 'No'. However, of the 121 people who actually made real use of Condor, *not a single one* answered 'No'. Of course, 18 real users remained with an answer of 'Not sure', which encouraged us to make a better effort to communicate with our users.

One might suggest that those who objected to data collection and mapping did not use Condor because of that objection. This is unlikely. Many users wrote comments in the free answer section describing why they did not use the software; some were just curious, some required different



platforms, some did not like the user interface and so forth. Not one cited the data collection issue as a reason for not using it.

Of course, a lack of 'No' answers does not indicate that all users are participating in data collection. As explained above, users can and do disable Condor's reporting features. One user explained as follows:

*I like your map, but am opting out at this time . . . Our legal guys are a little goofy about this stuff right now. I'll bet once Grids are more widely used, we'll opt back in.*

Although it is difficult to generalize user attitudes from a small survey, these data (and the quote above) are consistent with the following hypothesis: *familiarity breeds comfort*. Users that understand the nature and purpose of data collection are more likely to participate.

These questions also offer lessons in how (not) to design surveys. While writing question 6, we assumed that we should include the 'Don't have' option because the question only applied to those with Condor pools. However, the use of 'have' likely resulted in confusion. A user, installer or administrator might not formally own the machines on which Condor runs and thus answer 'Don't have'. Conversely, someone who downloaded Condor but did not use it might express an opinion regardless. It would have been better to omit the option and separate out responses via correlation afterward.

Finally, in order to better understand the coverage of the automatic measurements, we asked users to share details of their Condor pools. In the following question, the bold numbers and percentages indicate how many people entered an answer to each field.

**7—Please tell us about the Condor pool that is most important to your work. You may share as little or as much information as you are comfortable with.**

Condor central manager	<b>88</b>	31%
Condor pool total size	<b>137</b>	49%
Condor software version	<b>150</b>	54%
Most common operating sys	<b>177</b>	63%
Name of organization	<b>144</b>	51%
City	<b>153</b>	55%
Country	<b>169</b>	61%

Of the 88 pools reported via the survey, 81 were unique. In each case, there were two or three people with some administrative responsibility for the pool. Each responded from a different IP address and wrote substantially different comments in the free-form response box. As can be seen, the central manager (matchmaker) had the lowest response rate. This could be partially due to technical reasons; many respondents might not have known this by memory or have been willing to look it up in the system. It is also reasonable to assume that this piece of the data is the most security sensitive.

## 7. DATA OVERLAP

In 2004, we had timely data regarding running Condor pools from three distinct sources: e-mail updates, UDP updates and the user survey. It is useful to consider how these data collection methods overlap in order to understand their strengths, weaknesses and biases.



Table II. Pool properties by reporting method on 1 January 2004.

Update method	Number of pools	Number of CPUs	Average pool size	Invalid name	Matchmaker platform			
					Linux	Windows	SunOS	Other
Total	954	36 310	38	22%	60%	24%	10%	4%
E-mail	554	25 367	45	11%	68%	11%	14%	5%
UDP	691	24 657	35	22%	55%	29%	10%	4%
Survey	81	7560	93	24%	66%	27%	6%	0%

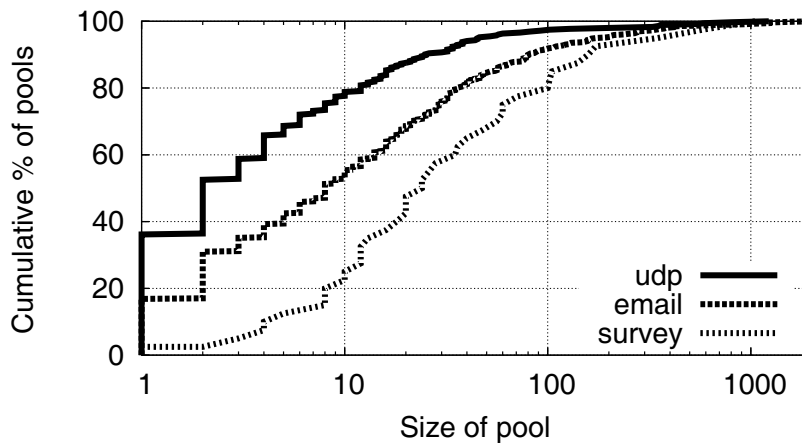


Figure 18. Cumulative distribution of pool sizes by collection method. The distribution of pool size by data collection method. E-mail tends to measure larger pools than UDP, while the survey tends to measure larger pools than e-mail.

Table II shows the properties of pools discovered by each reporting method on 1 January 2004. As can be seen, each reporting method emphasizes a different aspect of the system population. More pools were discovered by UDP (691) than by e-mail (554) or the survey (81), but a slightly larger number of CPUs were collected by e-mail (25376) than by UDP (24657). Pools discovered by e-mail had half the rate of invalid names (i.e. invalid syntax or WHOIS record) as pools reported by UDP or the survey. It is especially interesting that the survey discovered pools that were more than twice as large (93 CPUs on average) than e-mail (45) or UDP (35). This is confirmed in Figure 18, which shows the distribution of pool sizes across collection methods.

We hypothesize that these differences are due to the fact that a larger system represents a greater investment and thus a greater security concern and a higher attention to detail. The larger a system, the more likely it will have a working e-mail system that requires some administrative attention. Also, the more likely it will be placed entirely behind a firewall and be left out of both e-mail and UDP reports.



Table III. Pool properties by matchmaker platform on 1 January 2004.

Matchmaker platform	Number of pools	Number of CPUs	Average pool size	Invalid name	Percentage updated by		
					E-mail	UDP	Survey
Total	954	36 310	38	22%	58%	37%	4%
Linux	574	18 210	31	13%	66%	29%	4%
Windows	236	6168	26	51%	26%	67%	5%
SunOS	104	9807	94	11%	78%	17%	3%
Other	40	2125	53	0%	70%	30%	0%

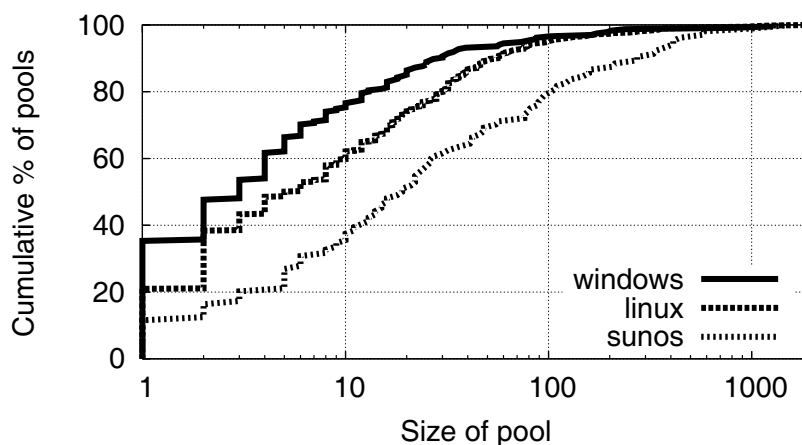


Figure 19. Cumulative distribution of pool sizes by matchmaker platform. The distribution of pool size by matchmaker platform. Note that all types have a large number of small pools and a small number of large pools. However, generally speaking, Linux pools are larger than Windows pools and SunOS pools are larger than Linux pools.

Furthermore, the larger a system, the more likely it is of strategic concern to its owners and thus the more likely they will respond to a survey.

It is also valuable to consider whether different reporting methods are more effective on different software platforms. As Table III shows, e-mail was much more effective than UDP on Unix-like platforms. More than twice as many pools were discovered by e-mail than by UDP on these platforms. However, the opposite holds on Windows: more than twice as many machines were discovered by UDP than by e-mail. In addition, there is a correlation between operating system and pool size. In general, Windows pools are smaller than Linux and Solaris pools. Figure 19 confirms this in the cumulative distribution. This is because large computing clusters are typically oriented toward Unix-based technical computing.

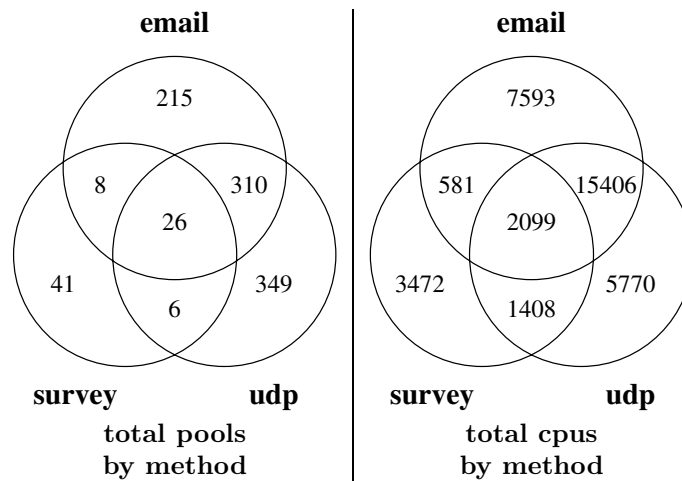


Figure 20. Overlap in coverage—1 January 2004. These Venn diagrams show the overlap in measurement techniques. Each circle represents pools or CPUs discovered by e-mail, UDP or survey. Note the large fraction of pools and CPUs discovered by exactly one technique.

Figure 20 emphasizes how disjoint the three measurement techniques are. The left Venn diagram shows how many pools were discovered by each combination of methods, while the right shows the total number of CPUs. In total, 608 pools (64%) and 17 135 CPUs (47%) were discovered by exactly one method with no duplication by other methods. This emphasizes that multiple techniques are necessary to capture the entire system. If we consider the number of pools detected by the survey, 40 were already known by e-mail and UDP updates, while 41 were unknown until the survey. This gives a very rough indication that only half of all operating pools are automatically reporting. However, it does not allow us to distinguish between automatic reports that are blocked by network conditions and those that have manually disabled.

Did the survey respondents tell the truth? We can gain some measure of this by comparing the cases where survey responses overlap with e-mail and UDP reports. Figure 21 shows the difference between 40 survey responses and the last data collected by automatic methods. Positive values indicate overestimates, while negative values indicate underestimates. Five responses gave correct answers, six underestimated and 29 overestimated, some by over 70 CPUs. On average, the respondents overestimated by 14%.

The respondents have little incentive to exaggerate, so we attribute the difference to the dynamic nature of Condor pools. If a startd is turned off or rebooted, the matchmaker will forget about it until it returns to life. Thus, any pool of sufficient size is likely to report a slightly smaller number of running startds than physical CPUs. It is reasonable to assume that respondents reported the tangible evidence of physical CPUs rather than the number of running startds. This result is more evidence in favor of our earlier hypothesis that a large fraction of machines are down at any given time.

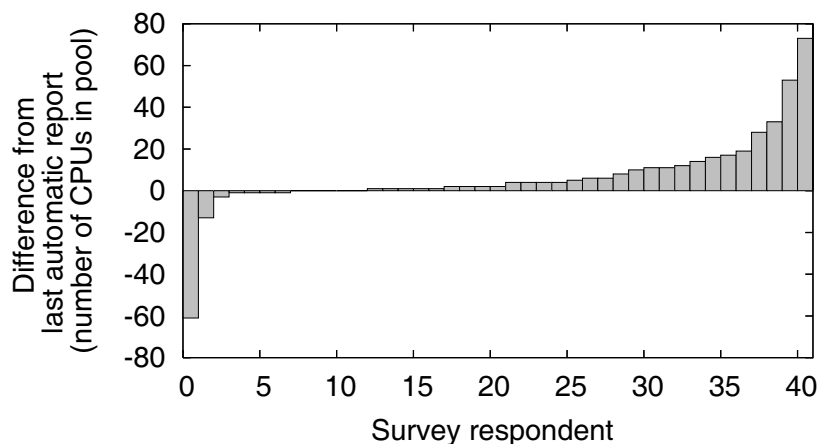


Figure 21. Survey accuracy. A comparison of surveyed pool sizes that could be verified by automatic reports. Positive numbers are overestimates, negative numbers are underestimates. Users generally overestimate the current pool size.

## 8. RELATED WORK

Although there are a wide variety of techniques for collecting data from running, large-scale distributed systems, they boil down to a few distinct philosophies. Online measurement of a system can be done by either pulling data, pushing data or by passive tracing. The choice between pulling and pushing data is a perennial concern in systems design. It is often presented as a performance issue [13,14] but in the case of a system with many autonomous participants, structural and social concerns are more important.

Many large-scale system studies have relied on pulling. For example, a large-scale study of the Andrew file system (AFS) [15] relied on a pull model. Every AFS server makes available a number of performance metrics that can be read via the `xstat` interface. Given a set of AFS servers, a data collector may poll each for data in order to discover a wide variety of statistics. Recursive pull models have been used to enumerate the domain name system (DNS) [16], to map peer-to-peer networks [17] and to index the Web [2].

The advantage of the pull model is that it allows for greater flexibility. Each node may record far more data than can easily be collected, allowing the collector to select from the available data at run time. However, as the system becomes larger, the data collector must be given a higher degree of parallelism and thus complexity. Furthermore, a pull model does not work well with systems that may include firewalls and other devices designed to limit incoming data. Finally, a pull requires that all participants either explicitly give access to the data collector or make their data publically readable.

Condor has relied primarily on a push model; every node sends self-contained messages on its own initiative. This has structural benefits; the senders of data can traverse firewalls, while the receiver of data can be a relatively simple event-driven process. Furthermore, by statically defining both the data



content and destination, we simplify the trust model (and our explanations) to end users. The primary drawback of the push model is that we must decide far in advance what data to collect.

If a system does not have a measurement structure built directly in, then traces must be manually attached to observe activity. This can only be done at large scale if there are significant organizational resources to apply to the problem. For example, the largest file system study undertaken so far was performed on 4801 personal computers at Microsoft Corp. [18]. As the authors note, the deployment of measurement software required the support of management to make sufficient notice across the company and provide incentives for users to participate. Like a survey, such a measurement must be careful to account for biases from selection of the participants.

Measurement can be done at even deeper levels of the system. For example, passive network tracing is commonly applied to distributed systems that have a central components such as file system servers [19,20]. However, this approach is difficult to scale up to very large systems. Network tracing requires establishing a high degree of trust between the network operator and the experimenter, not to mention non-trivial equipment to capture and record high-bandwidth data flows. In order to protect users of the network, traces must usually be anonymized, leaving little information to be gleaned about activities above the network level. For example, a large-scale network trace is the NLANR PMA packet header trace, which includes 14 sites in the United States research network infrastructure over a period of several years [21]. This trace almost certainly contains packets relevant to Condor, but without the *body* of the packets, there is little that can be inferred.

## 9. CONCLUSION

We have three pieces of advice to offer to those considering measuring large distributed systems.

### **Plan for measurement far in advance**

Adding a measurement infrastructure to a system *after* it has an installed a base and a user community is a very slow process. Over the last year, we have added a small number of additional fields to the periodic e-mail and UDP messages in order to report several more items of interested. However, as the version deployment results have shown, we will have to wait several years before these measurements will be taken on any significant fraction of Condor worldwide. We strongly recommended that system designers should include a measurement infrastructure in the earliest stages of design and implementation.

Of course this means that one must make guesses about what data to report. With hindsight, we should have included some information about recent failure (or at least system uptime) in each e-mail and UDP update. This would have allowed us to better understand the losses described in Section 6.1.

### **Use multiple techniques to collect data**

As we have shown, each technique for collecting data from a system has its biases. Without the benefit of multiple sources, you cannot reason about the bias of a single source. Furthermore, each technique has natural predators; e-mail is reduced by spam-filtering techniques, while UDP is reduced by firewalls

---



and network outages. It is difficult to predict which techniques will survive as the Internet evolves, so one must have multiple tools.

To this end, we would like to create a third automatic collection technique: a direct TCP connection from each matchmaker to the global collector. We believe that this technique would be more reliable than UDP and yet not require the same complex infrastructure as e-mail, thus covering a middle ground. However, this would require a greater attention to scalability in the global collector.

### Give the users a stake in measurement

Our overall experience is that users are far more amenable to the data collection when they have some understanding of its content and connection to its use. Anecdotally, users that learned about our data collection by seeing various maps and charts had a very positive response and would often even act to ensure that their systems were included. Users that learned about data collection from their own network traces naturally responded negatively.

One way for us to give the users a greater stake in Condor would be to establish a continuously running interactive survey that displays the set of Condor pools known worldwide, while allowing pool owners to add or delete their systems from the display as they see fit.

Current maps and data may be found at <http://www.cs.wisc.edu/condor/map>. Any user or administrator of a Condor system with concerns or questions about this data may contact us at [condor-admin@cs.wisc.edu](mailto:condor-admin@cs.wisc.edu).

### REFERENCES

1. Sullivan WT, Werthimer D, Bowyer S, Cobb J, Gedye D, Anderson D. A new major SETI project based on project serendip data and 100 000 personal computers. *Proceedings of the 5th International Conference on Bioastronomy*, 1997.
2. Brin S, Page L. The anatomy of a large scale hypertextual search engine. *Computer Networks and ISDN Systems* 1998; **30**(1-7):107-117.
3. Litzkow M, Livny M, Mutka M. Condor—a hunter of idle workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988. IEEE Computer Society Press: Los Alamitos, CA, 1988.
4. Thain D, Tannenbaum T, Livny M. Condor and the Grid. *Grid Computing: Making the Global Infrastructure a Reality*, Berman F, Fox G, Hey T (eds.). Wiley: New York, 2003.
5. Anstreicher K, Brixius N, Goux J-P, Linderoth JT. Solving large quadratic assignment problems on computational Grids. *Mathematical Programming, Series B* 2002; **91**:563-588.
6. Periakaruppan R, Nemeth E. Gtrace—a graphical traceroute tool. *Proceedings of the USENIX 13th Systems Administration Conference (LISA)*, November 1999. USENIX Association: Berkeley, CA, 1999; 69-78.
7. Daigle L. WHOIS protocol specification. Internet Engineering Task Force Request for Comments 3912, September 2004.
8. U.S. Federal Trade Commission. The integrity and accuracy of the WHOIS database. Testimony before the U.S. House of Representatives. <http://www.ftc.gov/os/2002/05/whois.htm> [May 2002].
9. Internet Corporation for Assigned Names and Numbers. WHOIS recommendation of the security and stability advisory committee SAC-003.2. <http://www.icann.org/committees/security/sac003.htm> [February 2003].
10. Association for Computing Machinery. WHOIS letter to ICANN. <http://www.thepublicvoice.org/news/whoisletter.html> [October 2003].
11. Electronic Frontier Foundation. Comments to ICANN's WHOIS task forces 1 and 2. [http://www.eff.org/Infra-structure/DNS\\_control/icann\\_whois.php](http://www.eff.org/Infra-structure/DNS_control/icann_whois.php) [July 2004].
12. Bono M. Securely Protect Yourself Against Cyber Trespass Act (SPY-ACT). *United States House of Representatives Bill H.R. 2929.EH*, October 2004.
13. Gwertzman J, Seltzer M. The case for geographical push-caching. *Proceedings of the 5th Workshop of Hot Topics in Operating Systems*, 1995. USENIX Association: Berkeley, CA, 1995.
14. Acharya S, Franklin M, Zdonik S. Balancing push and pull for data broadcast. *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '97)*, 1997. ACM Press: New York, 1997.
15. Spasojevic M, Satyanarayanan M. An empirical study of a wide-area distributed file system. *ACM Transactions on Computer Systems* 1996; **14**(2):200-222.



16. Ganatra NK. Census: Collecting host information on a wide area network. *Technical Report UCSC-CRL-92-34*, UC-Santa Cruz, Computer Science Department, 1992.
17. Ripeanu M, Foster I, Iamnitchi A. Mapping the gnutella network: Properties of large scale peer-to-peer systems and implications for systems design. *IEEE Internet Computing* 2002; **6**(1):50–57.
18. Douceur J, Bolovsky W. A large scale study of file-system contents. *Proceedings of the Conference on Measurement and Modeling of Computer Systems*, Atlanta, GA, May 1999. ACM Press: New York, 1999.
19. Blaze MA. NFS tracing by passive network monitoring. *Proceedings of the 1992 USENIX Winter Conference*. USENIX Association: Berkeley, CA, 1992.
20. Ellard D, Ledlie J, Malkani P, Seltzer M. Passive NFS tracing of email and research workloads. *Proceedings of the 2003 USENIX File and Storage Technologies Conference*. USENIX Association: Berkeley, CA, 2003.
21. Fomenkov M, Keys K, Moore D, Claffy K. Longitudinal study of internet traffic from 1998 to 2003. *Proceedings of the Winter International Symposium on Information and Communication Technologies*, January 2004. ACM Press: New York, 2004.