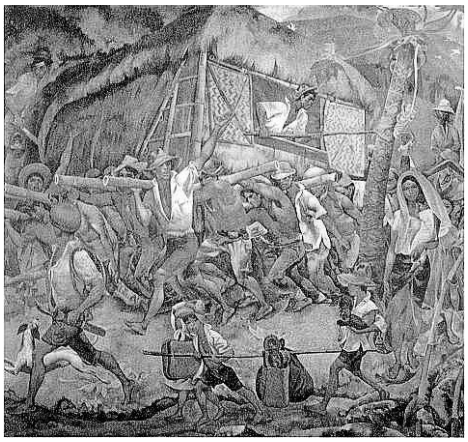
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING
AND COMPUTER SCIENCE

PH.D. THESIS PROPOSAL

VOLUNTEER COMPUTING

ACHIEVING ELECTRONIC BAYANIHAN ON THE INTERNET

by Luis F. G. Sarmenta
draft 1.0: *December 9, 1998*



INTRODUCTION

In the old villages of the Philippines, when a family moved to a new place, they not only took the kitchen sink with them, they took the entire house. Traditional Filipino houses were built with bamboo and thatch, and raised a few feet from the ground on stilt-like poles. These houses were light enough so that if a family needed to move to a new place, it was actually more economical for them to move the entire house to its new site rather than build a new house. Obviously, though, they could not do this by themselves, so their neighbors would get together and help them. They began by placing long bamboo poles length-wise and cross-wise under the house. Once this framework was in place, volunteers from the neighborhood gathered under it, and carried the whole house together on their backs all the way to its new site. Everyone happily did his or her own part—from the framework builders, to the house carriers, to the people on the roof pushing tree branches out of the way, and the ones preparing food and refreshments for everyone. Even children took part by carrying small items. At the end of the day, everyone gathered at the new site for a small neighborhood *fiesta* to celebrate their success.

This spirit of communal unity and cooperation, of people helping each other and working together, is called *bayanihan*.[1] Like its counterparts in other cultures, including *gotong royong* in Malaysia and Indonesia, *harambee* in Africa, and *barn raising* in the United States, *bayanihan* makes it possible to achieve seemingly impossible

[1] Pronounced like "buy-uh-**nee**-hun" with the stress on the third syllable. The image shown above is from "Bayanihan", a mural by Philippine National Artist Carlos "Botong" Francisco, commissioned by Unilab, Philippines in 1962. It is used here with permission from Unilab.

feats through the concerted effort of many people with a common goal and a sense of unity.

My Ph.D. thesis project, *Project Bayanihan*, seeks to bring the *bayanihan* spirit into the realm of world-wide computing by studying the idea of *volunteer computing*. Volunteer computing makes it possible to solve hard computational problems by allowing people around the world to easily pool their existing computing resources, and work together in a concerted effort. In this proposal, I present the idea of volunteer computing, discuss its many potential benefits, and identify research issues that must be addressed in implementing it. Then, I describe how I plan to explore these many issues in my thesis, and develop a prototype system that will allow people to start using volunteer computing for real applications, as well as serve as substantial groundwork for further research.

BACKGROUND AND MOTIVATIONS

PARALLEL PROCESSING, NOWS, AND METACOMPUTING

Volunteer computing is based on the idea of *parallel processing* – a large computational problem that takes a long time to solve on a single computer can be solved up to N times faster by first cutting it up into smaller independent parts, and then having N computers work in parallel on different parts of the problem at the same time.

Today's supercomputers work in this manner. A supercomputer typically contains tens, hundreds, or even thousands of interconnected processors. The combined power of all these processors working in parallel makes it possible to solve in a few minutes or hours computationally-intensive problems that would take days, months, or years to solve on a normal computer. Examples of such problems include high-energy physics simulations, global climate modeling and simulations, molecular structure analysis for chemical and biological research, data mining (i.e., searching for useful patterns in large sets of data) for business applications, and many others. Unfortunately, although they are fast, supercomputers are also extremely expensive, typically costing millions of dollars each, not including the expensive maintenance, software, and support costs due to the use of specialized processors, interconnections, and hardware. For this reason, only large government- or industry-funded research labs are currently able to afford supercomputers.

In response to this problem, several research groups in the past decade have developed systems that allow ordinary PCs and workstations on a network to be used in parallel as a virtual supercomputer. With these systems, universities and companies can harness the power of existing machines on their networks, and enjoy

the benefits of supercomputing at little or no extra cost. Often, these systems took advantage of idle cycles – times when a computer is not being used, either because the user is not there (e.g., at night), or the user is not performing any computations (e.g., the user is visually reading a web page). Such systems, sometimes called NOWs (networks-of-workstations), or SCANs (supercomputers-at-night), have successfully been installed and used in many research labs and universities, and several standardized programming systems for them already exist.

In the past five years, the explosive growth in the number of computers connected to the Internet has sparked a movement towards extending the range of NOWs and SCANs beyond the boundaries of individual organizations, and allowing people to pool their computing resources into global *networks of networks* of computers. In 1994, for example, a world-wide network of 1,600 computers succeeded in solving a 17-year old cryptographic challenge by factoring a 129-digit number [24]. Today, the Great Internet Mersenne Prime Search (GIMPS) [39], is using a similar but larger network to expand the range of known prime numbers. At present, several research groups have started working on generalizing the principles behind these ad hoc systems, and developing general-purpose *metacomputing* systems [27,16]. Their goal is to eventually allow people to use global networks of networks of computers as one big virtual *metacomputer*.

VOLUNTEER COMPUTING

Volunteer computing is a form of metacomputing that maximizes the ease with which people can have their machines be part of a metacomputer. Most existing NOW and metacomputing systems, while very flexible and powerful, require significant human effort to set up. First, programmers need to take into account the different types of computers (PCs, Macintoshes, UNIX workstations, etc.) in the metacomputing network, and generate versions of their programs for each distinct type. Then, the system administrators of the participating machines have to install these programs on all machines under their respective domains. Finally, the administrators of all participating domains must coordinate and give each other access rights to their machines, as required by the metacomputing software. All this means that it can take days, weeks, or even months to set up a large metacomputing network – that is, if it can be set up at all. The idea behind volunteer computing is to eliminate these setup requirements, and make it so easy to join a computing network that *anyone*, even a casual computer user without any technical knowledge, can do it. In this way, we not only make it possible to build much larger metacomputers much more quickly, but also make metacomputing and all its benefits more accessible to the common Internet user.

We can catch a glimpse of the enormous potentials of volunteer computing from the recent success of `distributed.net`, a cooperative group of programmers

and thousands of volunteers around the world who, on October 19, 1997, solved the $10,000 RSA RC-56 decryption challenge after trying 34 quadrillion different keys [5]. At the time the search ended, `distributed.net` had over 4,000 active volunteer teams (having anywhere from one to hundreds of computers each), and was searching through keys at a rate of over 7 billion per second – equivalent to the combined power of about 26,000 high-end PCs. Over the span of the 250 days it took to crack the code, they received results from as many as 500,000 different computers (unique network addresses) around the world [11].

A large part of the success of this project can be attributed to the relative ease with which users were able to participate in the computation. The programming team at `distributed.net` wrote versions of their programs for almost all the different processor and operating system combinations in use today. They also set up a number of server machines that automatically received results from volunteers through the Internet. With these in place, all that volunteers needed to do to join was to download the appropriate version of the software, install it on their machines, and then run the software in the background. There was no need for `distributed.net` administrators to personally install software on the volunteer machines and no need for volunteers to coordinate with each other beforehand. (That is, the process was easy enough that the volunteers could do it themselves, and the software did not require explicitly giving out access rights.)

Recent advances in World Wide Web technology offer to make volunteer computing even easier. Key among these is Sun Microsystems' Java technology [37], which allows people to write platform-independent programs called *applets*, that can be placed on a web page together with ordinary text and images. When a user views a web page, any applets on the page are automatically downloaded by the user's browser and executed on the user's machine, where they can provide dynamic and interactive features to the web page not possible with ordinary text and images. Java applets can be used to enhance web pages with such features as animations, interactive demos, and user-friendly input forms. Java's built-in networking features also allow applets to be used for interactive multi-user applications that allow people to interact with each other and share ideas through the Web, such as games, "chat rooms", and "shared whiteboards". Java's versatility has made it one of the most popular and fastest-growing software technologies today. All major web browsers now support Java, and thousands of individuals, companies, and schools are now using Java for a wide variety of personal, commercial, and educational purposes. (The Gamelan Java repository web site [13] contains a good collection of these applets.)

Java's applet technology, coupled with its popularity and ubiquity, offers us an opportunity to make volunteer computing both easier and more accessible than in systems like `distributed.net`. With Java, programmers who want to set up a

metacomputing network can write their software as Java applets and post them on a web page. Then, people who want to volunteer need only view the web page with their browsers, and wait. The applets are downloaded automatically and run without any human intervention. The volunteers do not even have to install any software themselves. In this way, adding a computer to a metacomputing network – something that used to require expert administrators and take days of setup time in conventional NOWs and metacomputers – can be done by the average user with literally a single mouse click.

The implications and potentials of this technology are enormous. By making it easy for every one of the world's millions of computer users, experts and casual users alike, to have their computers work together, volunteer computing creates many new technological and social possibilities. Project Bayanihan seeks to identify these and to develop theories and technologies to turn them into reality.

POTENTIAL BENEFITS AND APPLICATIONS

The potential benefits of volunteer computing range from the local to the global. Volunteer computing can be used within and between organizations not only to provide inexpensive supercomputing capabilities, but also to facilitate and encourage global collaboration and sharing of resources. With the appropriate economic models and mechanisms, it can also be used in commercial systems where computing power becomes a commodity that people can buy, sell, or trade. Finally, volunteer computing has the potential capability to harness the millions of the computers on the Internet, and use them towards solving hard computational problems for the common good of the whole world.

INTRA-ORGANIZATION VOLUNTEER COMPUTING

Many companies and universities today have internal networks (intranets) with many PCs and workstations that remain idle most of the time – not only during off-hours, but also during office hours when they are mostly used for non-computational tasks such as word processing. Volunteer computing can be used to pool together the computing power of these existing and under-utilized resources to attain supercomputing power that would otherwise be unaffordable. This not only makes it possible for research organizations to satisfy their computational needs inexpensively, but also creates an opportunity for other organizations to consider the use of computational solutions and tools where they haven't done so before.

For example, companies with heavy computational needs can use volunteer computing systems as an inexpensive alternative or supplement to supercomputers. Some applications include: *physical simulations and modeling* (e.g., airflow simulation in

aircraft companies, crash simulation in car companies, structural analysis in construction firms, chemical modeling in chemical, biomedical, and pharmaceutical labs, etc.), *intensive data analysis and data mining* (e.g., for biomedical labs involved in genome research, for financial analysts and consulting firms, etc.), and *high-quality 3D graphics and animations* (e.g., for media and advertising companies).

At the same time, companies that have hitherto deemed computational solutions or tools too expensive can start benefiting from them. For example, manufacturing companies can benefit from computational analysis and *process simulations* of their complex pipelines. These can be used to expose bottlenecks and weak points in processes, predict the effects of changes, and lead to finding ways to make them more efficient and cost-effective. Marketing departments of companies can do *data mining* on sales and consumer data, finding patterns that can help them formulate strategies in production and advertising. In general, by providing an affordable way to do computation-intensive analysis, volunteer computing can let companies enhance the intuitive analysis and ad hoc methods currently used in planning and decision making, and make them more informed and systematic.

Like companies, universities and research labs can use volunteer computing to turn their existing networks of workstations into virtual supercomputers that they can use for their research. This is especially useful in financially-constrained institutions, such as universities in developing countries that cannot afford to buy supercomputers and have thus far been unable to even consider doing computation-intensive research. Even in non-research-oriented universities that do not have a true need for supercomputing, virtual supercomputers built through volunteer computing can be used to *teach* supercomputing techniques, giving students skills they can use when they go to graduate school or industry.

Most of the applications mentioned here can be implemented with conventional NOW or SCAN software, and are in fact already being implemented in a growing number of organizations today. The advantage of volunteer computing over these is that: 1) it makes implementing applications dramatically easier for administrators and users alike, and 2) by doing so, it makes the technology accessible to many more organizations, including those that do not have the time and expertise to use conventional NOWs. For example, if a company decides to use its machines for parallel processing, system administrators do not need to spend time manually installing software on all the company machines anymore. They can simply post the appropriate Java applets on the company web site, and then tell their employees to point their browsers to a certain web page. The employees do so, and leave their machines running before they go home at night. In this way, a company-wide volunteer computing network can be set up literally overnight, instead of taking days or weeks for installing software and educating users as it would in conventional systems.

INTER-ORGANIZATION VOLUNTEER COMPUTING

The same mechanisms that make volunteer computing within individual organizations work can be used to make volunteer computing between organizations work as well. By volunteering their computing resources *to each other*, or to a common pool, organizations around the world can share their computing resources, making new forms of world-wide collaborative research possible.

Much research today is going into developing technologies for *collaboratories* – world-wide virtual laboratories where researchers in different labs around the world can interact freely with each other and share data and ideas, as if they were all in one big laboratory [41,28]. Technologies currently under development include videoconferencing, shared whiteboards, remote instrument control, and shared databases. Volunteer computing can enhance collaboratories by allowing researchers to share not only data and ideas, but computing power as well.

Even organizations that do not collaborate very closely can use volunteer computing mechanisms to *barter trade* their resources, depending on their need. This has interesting geographical possibilities. For example, a university in the United States can allow a lab in Japan to use its CPUs at night (when it is day in Japan) in exchange for being able to use the Japanese lab's CPUs during the day (when it is night in Japan). In this way, we get a win-win situation where both labs get twice the processing power when they need it.

Taken to an extreme, pooling and barter trade of computing resources between organizations can lead to the formation of *cycle pools* – massive pools of processing power to which people can contribute idle cycles, and from which people can tap needed processing cycles. This can be used for organizations' individual gains, as in the example above, but can also be a venue for larger organizations to help smaller ones and contribute toward a better social balance of resources. For example, a large university with more idle computer power than it needs can donate its idle cycles to the pool, and allow smaller universities and schools to make use of them for their own teaching and research needs.

COMMERCIAL SYSTEMS

By giving people the ability to trade computing resources depending on their needs, volunteer computing effectively turns processing power into a commodity. With appropriate and reliable mechanisms for electronic currency, accounting, and brokering, *market systems* become possible, allowing people and groups to buy, sell, and trade computing power. Companies needing extra processing power for simulations, for example, can contact a broker machine to purchase processing power. The broker would then get the desired processing power from a cycle pool

formed by companies selling their extra idle cycles. Even individuals may be allowed to buy and sell computing power.

Commercial applications of volunteer computing also include *contract systems*, where computing power is used as payment for goods and services received by users. For example, information providers such as search engines, news sites, and shareware sites, might require their users to run a Java applet in the background while they sit idle reading through an article or downloading a file. Such terms can be represented as a two-sided contract that both sides should find acceptable. For example, while it is actually possible in today's browsers to hide Java applets inside web pages so that they run without users knowing about them, most users will consider this ethically unacceptable, and possibly even a form of theft (of computing power). Sites that require users to volunteer must say so clearly, and allow users to back-out if they do not agree to the terms.

If appropriate economic models and mechanisms are developed, commercial volunteer computing systems can allow people not only to *save* money as they can in organization-based systems, but to *make* money as well. Such an incentive can attract a lot of attention, participation, and support from industry for volunteer computing. In the end, what happened to the Internet may happen to volunteer computing as well – it will start out as something primarily used in academia, and then when it becomes mature, it will be adopted by industry, which will profit immensely from it.

NOIAS

Many experts, both in industry and in the academic community, predict that in the near future, *information appliances* – devices for retrieving information from the Internet which are as easy to use as everyday appliances such as TVs and VCRs – will become commonplace. In the United States today, companies such as WebTV [40] are starting to develop and sell information appliances in the form of "set-top boxes", while cable companies such as MediaOne [25] are starting to support high-speed cable modems that use the same cable that carries the TV signals to connect users to the Internet up to 50 times faster than a telephone modem can. It is not hard to imagine that within the next five or ten years, the information appliance will be as commonplace as the VCR, and high-speed Internet access as commonplace as cable TV.

This brings up an interesting idea: what if we use volunteer computing to allow users of these information appliances to volunteer their appliances' idle cycles? These nodes can perform computation, for example, when the user is not using the information appliance, or while the user is reading a web page. Such networks-of-information-appliances, or NOIAs, as I call them (incidentally, the acronym also

means "mind" in Greek [νοια], and may evoke an image of a brain-like massively parallel network of millions of small processors around the world), have the potential for being the most powerful supercomputers in the world since: 1) they can have tens of millions of processors (i.e., potentially as many as the number of people with cable TV), and 2) although they are only active when the user is idle, we can expect the user to be idle most of the time.

NOIAs can be contract systems. Cable companies or Internet service providers (ISPs) can sign a contract with their clients that would require clients to leave their information appliance boxes on and connected to the network 24 hours a day, running Java applets in the background when the user is idle. This may be acceptable to many clients since most people today are used to leaving their VCR on 24 hours a day anyway. In some cases, however, clients may object to having someone else use their appliances when they are not using it, so a reasonable compromise may be to always give clients the option of not participating, but give benefits such as discounts or premium services to clients who do. In addition, volunteering clients may be allowed to indicate which kinds of computations they do and do not want to be involved in. For example, a client may allow her appliance to be used for biomedical research directed at lung cancer, but not for tobacco companies doing data mining to improve their advertising strategy.

It may take some time before information appliances and high-speed Internet access become widely available enough to make NOIAs possible. However, it is useful to keep them in consideration since techniques developed for the other forms of volunteer computing are likely to be applicable to NOIAs when their time comes.

TRUE VOLUNTEER COMPUTING

Finally, and perhaps most interesting of all, there are *true* volunteer computing systems, where the participants are volunteers in the true sense of the word. That is, they: 1) come and leave of their own free will, 2) are unknown to the administrators before they volunteer – i.e., they can come from anywhere, not just from the administrators' domains, and 3) do not expect substantial compensation.

By allowing anyone on the Internet to join as they please, true volunteer computing systems have the potential for pooling together the computing power of the millions of computers on the Internet. Today, it is estimated that there are about 30 million computers on the Internet. If we can somehow get even a small fraction of these to work together, we would have a world-wide supercomputer more powerful than any single supercomputer by several orders of magnitude. The experience of `distributed.net`, which managed to involve as many as half a million computers over 250 days in cracking the RC5-56 challenge despite requiring some technical knowledge on the part of volunteers, shows that this idea is not as

far-fetched as it may sound at first. Because it lets users volunteer through the web using just their browsers, Java-based volunteer computing systems can be even larger than `distributed.net`, and can be formed much more quickly.

The experiences of popular web sites have shown that it is possible to attract as many as thousands, or even millions of users in a matter of hours. In July 1997, for example, the NASA Mars Pathfinder web sites received 46 million "hits" per day as people came to view images from Mars [26]. Since each visit from a user actually consists of several hits, this does not mean 46 million unique users. Nevertheless, it represents at least tens and probably hundreds of thousands of users, and earned the NASA web site a place in the 1997 Guinness Book of World Records. Digital's AltaVista, a popular web search engine, may be even more impressive. It has been serving 20 million searches per day and 18 million unique users per month for the past several months [10]. All these examples show that ultimately, the size and speed of a Java-based true volunteer computing system are limited only by how well it can attract people, and how many volunteers it is prepared to handle.

This leads to the question of what kinds of applications can attract a large number of volunteers. To date, the most popular and successful applications have been what we might call "cool" challenges – challenging computational problems that people join just for fun and the pride of being part of a global effort. So far, these have mostly involved mathematical problems such as searching for prime numbers (e.g., GIMPS), and cryptographic challenges (e.g., the 1994 factoring effort, and `distributed.net`). Other possibly more interesting cool challenges include: *chess* (imagine a "Kasparov vs. the World match", where Kasparov plays against a world-wide volunteer computing network), *international computer olympics* (countries form their own volunteer computing networks, which then play against each other in competitions, such as chess, where the winner is determined by a combination of algorithm design and the number of loyal citizens willing to participate), and *distributed graphics applications* ("Join in the making of Toy Story III!").

Scientific problems in areas that appeal to the public can also attract volunteers. Recently, the SETI@home project [34], has started soliciting for volunteers willing to allow their home computers' idle cycles to be used for analyzing radio telescope data for unusual patterns that may indicate extraterrestrial life. With the help of publicity from TV and from the movie *Contact*, SETI@home attracted 35,000 potential volunteers within three months. (These are people who expressed willingness to participate; the actual computation has not started yet.)

Ultimately, true volunteer computing can be used for *worthy causes* – practical, useful, and relevant problems that serve the common good. Worthy causes can be local or global. *Local worthy causes* are those relevant to a particular city, country, or geographic area. These can include traffic simulation studies for congested mega-

cities, and local weather and seismological simulation research. *Global worthy causes* have a wider scope, and include those that can help humankind as a whole, such as molecular modeling and simulations for vaccine and pharmaceutical research, and simulating the effects of environmental policies on global warming and other environmental effects.

With true volunteer computing, an organization with a worthy cause can invite concerned Internet users to donate their idle cycles by simply visiting a web page and leaving their browsers running in the background while their work. Volunteers contribute processing power towards the worthy cause willingly, and according to their own capacity. The end result is electronic *bayanihan* – people working together for the good of their community and even the world.

RESEARCH ISSUES AND CHALLENGES

Making volunteer computing into a reality involves many interesting and challenging questions, issues, and problems. These include technical issues that need to be addressed in making volunteer computing possible and effective, economic issues, and social issues concerned with how this technology can and will affect society.

TECHNICAL ISSUES

The technical issues can be classified broadly into accessibility, programmability, reliability, and scalability and performance.

ACCESSIBILITY

Since the goal in volunteer computing is to potentially allow *anyone* on the Internet to participate, volunteer computing systems must be *easy to use*, and *platform-independent*. Volunteering must require as little technical knowledge from volunteers as possible. Even a seemingly simple procedure such as downloading and installing a program may be too complex, since most computer users today generally only know how to use applications, not how to install them. At the same time, users should be able to participate regardless of what type of machine and operating system they use, and preferably without having to identify their platform type.

In order not to discourage people from joining, volunteer computing systems must also be *secure*. Since programs will be executed on the volunteers' machines, volunteers must be given the assurance that these programs will not do harm to their machines. In current NOW and metacomputing systems, as well as volunteer systems such as `distributed.net`, programs run in user-mode, which means that they can access any of their users' files and data, including sensitive data such as

credit card information and passwords. Clearly, such a serious security risk would (and should) make people wary of volunteering and thus poses a major obstacle in achieving volunteer computing networks of larger sizes. In fact, in response to some incidents with Trojan horses created by hackers, `distributed.net` has recently started posting warnings telling volunteers to make sure that they only download programs directly from `distributed.net`.

Finally, volunteer computing systems should have a *good user interface design* to encourage volunteers to stay. In most traditional parallel applications, user interfaces for the processing nodes are unnecessary because these nodes are usually hidden inside a large supercomputer and cannot be accessed independently anyway. In volunteer computing systems, however, volunteers need an interface for doing such things as starting and stopping applets, or setting their priority. They also need some sort of progress indicator to assure them that their machines are actually doing useful work. User interfaces for viewing results and statistics, or for submitting jobs or specifying parameters, are also important. Whereas these are traditionally available only to the server's administrators, we may want to make them available to users as well. In commercial systems, for example, users would like a good interface for submitting problems, and receiving results.

Among technologies available today, Java satisfies these requirements best. As we have seen earlier, web-based systems using Java applets are both easy-to-use and platform-independent, allowing anyone with a Java-capable web browser (available on all major platforms today) to volunteer simply by visiting a web site. Furthermore, Java applets are executed in a secure "sandbox" that prevents them from accessing or destroying the user's data. Finally, Java provides a flexible graphical user interface (GUI) library that makes user interface design relatively easy. Other systems with similar features exist, and can also be used for volunteer computing, (e.g., MIT's Curl [21]). Currently, however, Java's popularity and ubiquitous presence make it the most practical choice.

PROGRAMMABILITY

Programming tools and libraries for volunteer computing must allow programmers to implement a wide variety of parallel applications, and be able to do so easily and quickly.

To make this possible, programming libraries and tools must be *platform-independent* so that programmers, like users, do not have to worry about the different platforms on which their programs may be run. The libraries should provide standard interfaces for network communications, graphical user interfaces, data packets, etc., such that programmers can just write one version of the program, and not have to modify it for each distinct platform. Libraries and tools should also

support *modularity* and *reusability*. Programmers should be able to reuse already-made parts in their programs, as well as create parts that can be reused in the future, or used by others. Ideally, parts in the software library should be like LEGO bricks – simple yet connectable, and consequently, highly versatile and reusable.

Again, Java is a good starting point in this regard. Aside from providing a platform-independent interface to the programmer, it is also *object-oriented*, encouraging (even forcing) programmers to program in a modular and reusable manner.

Programmers must also have ways to write parallel programs easily. While this is a problem in all parallel systems, including supercomputers, NOWs, and metacomputers, it is an even harder problem in volunteer computing, since volunteer nodes can have different kinds of CPUs, and can join and leave a computation at any time. For example, in traditional parallel programming, it is not uncommon to write an instruction that says something like, "At time 10, Processor 1 sends data *A* to Processor 2." This may not work in a volunteer computing system because Processor 1 may be a slow machine, and will not have data *A* ready by time 10, or worse, Processor 2 may leave the system, in which case Processor 1 is stuck with no one to send to. In general, parallel programming models for volunteer computing need to support *adaptive parallelism*. That is they must work without assuming the existence of a fixed number of nodes, or depending on any static timing information about the system. Various strategies for implementing adaptive parallelism such as Linda/Piranha [15], eager scheduling [4], and Cilk [6] are known, and can be starting points, but there is still a lot of room for improvement and further research.

RELIABILITY

Because of their size, volunteer computing systems are especially prone to faults. Not counting the problem of volunteers leaving (which is already covered by adaptive parallelism), faults can include not only *unintentional* random faults, such as data loss or corruption due to faulty network links or faulty processors, but also *intentional* faults caused by malicious nodes submitting erroneous data.

In general, *fault-tolerance* can only be achieved using some form of redundancy. For example, we may give the same piece of work to three different processors, and have them vote on the correct answer. Such *replication* techniques are used today in many critical systems, including satellites and spacecraft. Unfortunately, however, they are quite inefficient since repeating work *r* times means taking *r* times longer to solve the whole problem. Thus, an interesting research problem is to develop effective but efficient fault-tolerance techniques. One possible approach, for example, is *spot-checking* with *blacklisting*, where results are only double-checked

occasionally, but faulty nodes that are caught are not used again. This is less reliable than replication, but potentially much more efficient.

Commercial and true volunteer computing networks are also susceptible to *malicious* attacks from volunteers. One type of attack is *sabotage*, where volunteers (and possibly also non-volunteers) intentionally submit erroneous results. Instead of doing the computation it is asked to do, for example, a saboteur might simply return random numbers as results. Unchecked, such an error can propagate and render all future computations invalid – even those from honest volunteers. Another type of attack is *spying*, where volunteers steal sensitive information from the data they are given. Suppose, for example, that a company *A* purchases computing power from a commercial cycle pool in order to process its sales data. Without some form of protection, a competitor *B* can then spy on *A*'s data by joining the cycle pool and volunteering to do work.

Guarding against malicious attacks is a very challenging problem. It is also one that has not been studied very well since, so far, people have been able to trust the parts in their computer not to intentionally sabotage their computations. Possible ways of addressing this problem include using cryptographic techniques such as *digital signatures* and *checksums* to protect against volunteers sending back random results, *encrypted computation* to protect against spying and attempts to forge digital signatures and checksums, and *dynamic obfuscation* (i.e., instruction scrambling) techniques from computer virus research to simulate encrypted computation when it is not possible. These techniques are discussed in more detail in my paper [30], but remain to be implemented and tried.

In some cases, reliability problems can be addressed by *problem choice*. Some problems do not really require 100% accuracy, such as sound or graphics processing where a little static or a few scattered erroneous pixels would be unnoticeable. Others, such as search problems with rare results, have easily verifiable results that can be checked by the server without loss of performance.

Perhaps the most interesting problems, though, are those that use *self-correcting algorithms*, where erroneous results can at worst only delay the answer, not invalidate it. For example, some optimization problems can use so-called *genetic algorithms* [20] based on the principle of natural selection. These algorithms begin with a set of randomized solutions, and run for many cycles. In each cycle, a desirability function is used to screen out less desirable solutions. Those that remain are allowed to live and possibly "mate" with other solutions. Some are also randomly mutated to introduce variety and to prevent the population from getting stuck with a sub-optimal solution. After many cycles, the solution to the problem is taken from the set of "survivors". Volunteer computing systems employing genetic algorithms have a good chance of being resilient to faults and sabotage since any non-desirable

solutions returned by faulty processors or saboteurs will be screened-out by the desirability function. At worst, saboteurs can only make the algorithm take longer by introducing undesirable solutions into the system. At best, they may even inadvertently introduce benevolent mutations that will lead to the desired solutions more quickly.

SCALABILITY AND PERFORMANCE

To reach their full potential, volunteer computing systems must be *scalable*. That is, they must be capable of solving arbitrarily large problems by using arbitrarily large numbers of volunteers.

One obstacle to scalability is the need to have *servers* to distribute work, collect results, and coordinate the worker machines. Single-server systems are easy to set up and good for small systems but are eventually limited in size to the finite number of worker machines that the server can handle. For medium-sized systems such as inter-organization networks, a *pool* of servers may be used to support the bigger load, just like the NASA Pathfinder web site used "mirror sites" around the world to handle its load. For true volunteer systems that can potentially have millions of participants, however, even such server pools may not be sufficient. In such cases, a possible solution might be to employ *volunteer servers* to serve other volunteers. This is not easy, however. Not only does it involve challenging implementation problems, but it also raises more reliability questions such as what happens if the volunteer server crashes or leaves, or even what happens if the volunteer server is malicious.

Another problem related to scalability is the question of overall performance. Unlike supercomputers, which employ high-speed processors with high-speed interconnections, volunteer computing networks are composed of slower processors with much slower connections. (A modem line, for example, is thousands, or even millions, of times slower than the inter-processor links in today's supercomputers). So far, we have argued that this is not necessarily a problem since volunteer computing allows us to employ far many more processors than supercomputers do, and so presumably can make up for the difference by sheer numbers. Intuitively, this may sound plausible, but whether it is actually true remains a question.

To answer this question, it would be helpful not only to do experiments and make empirical observations, but to try to identify *theoretical* limits as well. In particular, it would be useful to derive a general set of equations relating parameters such as the total number of processors, the processing speed of each one, the processing speed of the servers, communication delays, problem granularity (i.e., computation/communication ratio), and other factors, in such a way that we can determine requirements for one parameter given the values of others. In addition, it

may also be useful to look into unconventional forms of computing for ways to get around these limits. After all, we know that the brain is able to achieve enormous processing capabilities by using very many very simple components (i.e., neurons). Can we do something similar with volunteer networks and NOIAs? Research on neural networks, cellular automata, and more recently, *amorphous computing* [1], which all use very small components to perform computation, may be starting points for this kind of theoretical research.

ECONOMIC ISSUES

MODELS AND MECHANISMS FOR COMMERCIAL SYSTEMS

In implementing commercial volunteer computing systems, we need to have *models* for the value of processing power. How does supply and demand affect the value of processing power? How about quality of service? How does one arrive at guidelines for fair barter trade? We also need *mechanisms* to implement these models. These include electronic currency mechanisms to allow money to be electronically exchanged in a safe and reliable manner, an accurate system for accounting of processing cycles, and efficient brokering mechanisms for managing the trading of processing cycles.

Because these mechanisms deal with real money, it is important that they are accurate and robust. Loopholes that permit crimes such as electronic forgery (of currency), and "cycle theft" to occur can lead not only to loss of computational power and data, but also to direct financial losses. Because of the difficulty in implementing these mechanisms, commercial volunteer computing systems based on them will probably not be possible in the near future, even if all the (non-economic) technical issues can be resolved. Nevertheless, some research groups, such as the SuperWeb group at the University of California at Santa Barbara (UCSB) [2], are already pursuing this goal by looking into the results of researchers in different but related areas, such as electronic commerce and digital cash. Meanwhile, it may also be worthwhile to look into implementing commercial systems with more relaxed mechanisms, such as barter trade, that may be less accurate, but at least cannot result in large direct financial losses.

HIDDEN COSTS

One of the early arguments for using NOWs (networks of workstations) instead of supercomputers was that of cost-effectiveness: whereas a single supercomputer costs millions of dollars, a NOW uses existing workstations and thus costs next to nothing. In this way, NOWs can bring parallel processing capabilities to people who cannot afford to buy supercomputers. When considering *massive* NOWs such

as volunteer computing networks or NOIAs, however, the validity of this argument may not be so clear anymore. Several issues arise and require further study.

For one, the cumulative cost of supplying electrical power to a massive NOW with thousands or millions of nodes may eventually offset the cost advantage gained from using existing hardware. Because of overhead, fault-tolerance redundancy requirements, and other factors, we expect that a massive NOW with the same processing power as a supercomputer would need several times as many nodes. Furthermore, each of these nodes would be an entire PC or workstation instead of just a single processor. Thus, it is easily conceivable that a NOW can use much more total power than a supercomputer.

We might argue that this power is already being spent by computers sitting idle, and that we are just putting it to better use. This may be true, but it deserves more study. It may be that with power-saving mechanisms, PCs and information appliances would spend less energy when idle. As a very simple example, when a user is not using his information appliance, he can turn it off. If we required the user to allow his information appliance to be used in a NOIA as part of his contract, then he cannot turn it off and it will be spending energy which he would not normally be spending otherwise.

Another source of hidden costs is the network. The high-speed links needed to alleviate congestion and improve performance in massive cycle pools may end-up costing more than a supercomputer. These links are often leased from a telephone company, and thus can be quite expensive. Furthermore, if a volunteer computation uses a public network such as the Internet, then moving data between computers can cause congestion in some areas, which can then affect unrelated Internet traffic, and cause equivalent financial losses in places uninvolved with the project.

I do not think that these problems would be enough to make volunteer computing completely untenable. However, I think that they are real concerns and should be studied.

SOCIAL ISSUES

Finally, volunteer computing raises up a number of social questions and possibilities. Among these are:

- How do you attract people to true volunteer computing efforts? What kinds of problems would people like to help solve? Would the offer of compensation attract more people? If so, what forms of compensation? (For example, `distributed.net` offered part of the $10,000 prize from RSA to the team who found the key).

- How will people react to NOIAs? Will they allow their computers to be used? Is it ethical to require them to participate as part of a contract? Will people see it as a form of charity and community spirit, or will they see it as a form of taxation? How can NOIAs be put to good use?

- In what ways will volunteer computing change and enhance global collaborative relationships? By allowing people to work together on a common goal, regardless of physical and political boundaries, can volunteer computing somehow foster global unity and peace? Is world-wide *bayanihan* really possible?

As we have seen, volunteer computing can offer us many new possibilities in computing. Ultimately, the basic social question is: how can we use volunteer computing to improve life on earth?

PROJECT BAYANIHAN: THE THESIS AND BEYOND

The ultimate goal of Project Bayanihan is to investigate all these technical, economic, and social issues, and develop useful theory and technology that will make volunteer computing a ubiquitous technology that will benefit the average computer user. Clearly, however, this goal cannot be achieved in one big step or in one Ph.D. thesis. There are enough issues, and enough different possible approaches to these issues, that the potential for research cannot be exhausted within a few years or by a few researchers. (In fact, several research groups have already been formed around the world that are dedicated to studying these issues, or even just subsets of these issues.)

In my thesis, therefore, I do *not* intend to exhaust the topic of volunteer computing. I do hope, however, to lay substantial groundwork that will enable and facilitate further research into volunteer computing by myself and by others. Specifically, my thesis objectives are:

1. to *identify* the potentials in volunteer computing, as well as the research challenges that need to be addressed in order to realize these potentials;

2. to develop a flexible *framework* that can be used not only to implement volunteer computing systems that can be applied to real computational problems, but also to explore different approaches to various research issues;

3. to use this framework to *initiate* exploration into the many technical issues we have identified (i.e., accessibility, programmability, reliability, scalability, and performance) by trying one or two specific approaches to each of them, observing the results, and identifying directions for future research; and

4. to produce a *software package* for public beta testing that will enable others to create volunteer computing systems that can be used for at least *some* classes of practical parallel problems within their organizations.

While time and resource constraints limit me to focusing primarily on *intra*-organization volunteer systems, I do hope to produce results that can be extended, by myself or by others, to implement *inter*-organization systems and true volunteer systems in the future. I do *not* expect to be able to explore the economic and social issues of volunteer computing myself in the near future, but I believe that in the time it will take to study inter-organization and true volunteer systems, others researchers in the electronic commerce and information technology industries will probably make significant developments technologies such as electronic currency and set-top boxes. When this happens, we can use their results in building commercial volunteer systems and NOIAs as well.

CURRENT WORK

I started work on Project Bayanihan in May 1996, first recognizing the potentials of Java-based volunteer computing, and identifying the many research challenges discussed earlier in this proposal. (These were also published in [30].) Since then, I have made significant progress towards my other thesis objectives as well, having successfully developed a flexible object-oriented framework, and used it to implement simple volunteer computing applications and experiment with different approaches to several technical issues. In the next few months, I plan to continue with these experiments as well as prepare my software for public use. I intend to finish my thesis in the Spring term of 1999.

THE BAYANIHAN SOFTWARE FRAMEWORK

I have developed an extensible object-oriented software *framework* that allows researchers to implement different ways of addressing technical issues such as adaptive parallelism, reliability, user-interface design, and scalability, while at the same time making it easy for programmers to set up Java-based volunteer computing systems for their own applications. This framework provides programmers with a set of basic components and an infrastructure for composing, interconnecting, and extending them. It uses the distributed object programming package HORB [19], to make programming easier by allowing programmers to concentrate on designing objects and specifying how they interact, rather than on low-level details such as data formats and communication protocols.[2]

[2] HORB is similar to Sun's standard Remote Method Invocation (RMI) package [27] in that it allows programmers to use objects distributed across several machines as if they were all on one machine.

Figure 1 shows an example of a volunteer computing system built using the Bayanihan framework. A Bayanihan system consists of many *clients* connected to a *server*. A *server* is typically a command-line Java application running on a machine with a web server for serving out Java applets. It contains one or more *problem* objects representing different parallel applications. Each problem has a *program* object that creates application-specific *data objects*, and places them in *data pools* managed by different *manager* objects. A *client* can either be a browser-based Java applet or a command-line Java application downloaded from the server and executed on a volunteer's machine. There can be different kinds of clients, each containing an *engine* object that communicates with a corresponding manager object by exchanging *data* objects (through an intermediate *advocate* object). A *worker client* for performing computation, for example, would have a *work engine* that would get and process *work data* objects from the *work manager*, while a *watcher client* for viewing results and statistics would have a *watch engine* that would get *result data* objects from a *watch manager*. Data objects are generally *polymorphic*, and know how to process themselves. A work object for a specific application, for example, would implement its own `doWork()` method, which the work engine then calls to run the computation. Both the engine and data objects have associated *GUI* (graphical user interface) objects that provide a user interface.

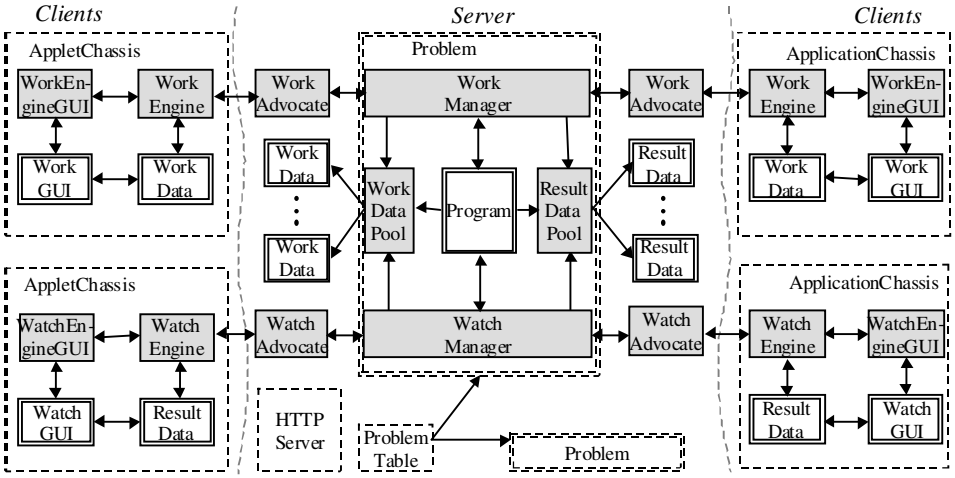


Figure 1: A Bayanihan system with generic (shaded) and application-specific (double-bordered) objects

Writing an application using the Bayanihan framework typically involves first selecting and using existing *generic* library components (shown shaded in Figure 1), and then defining new *application-specific* components (shown double-bordered) by extending existing base classes. In this way, application programmers can write a wide variety of applications that share a common programming model by using a common set of pre-defined engine, manager, and data pool objects, and then

defining different data, GUI, and program objects according to the application. At the same time, the framework also allows programmers and researchers to change the *generic* objects themselves, making it easy to implement and experiment with new generic functionality and mechanisms. For example, researchers can experiment with performance optimization by writing work manager objects with different scheduling algorithms. Similarly, replication-based fault-tolerance mechanisms can be implemented by extending (i.e., subclassing) the manager and data pool objects. Programmers can even implement entirely new parallel programming models by creating new *sets* of engines, managers, and data pools.

WRITING APPLICATIONS AND EXPLORING ISSUES

Using this framework, I (with the valuable advice and help of others) have successfully implemented a number of applications, as well as initiated explorations into various research issues. Some of our current results include:

- *Accessibility*. The Bayanihan software works with any web browser that can run Java 1.0 applets. No other software is required, and volunteers can join simply by clicking a link on a web page. For more complex applications, volunteers can also use Java *applications* which need to be downloaded and installed manually, but are have less file and network access restrictions than applets.

- *Programming Interface and Applications*. I have developed a set of generic components that support a simple *master-worker*-style programming model wherein the server divides a parallel problem into independent work packets which are then distributed to the volunteer workers [27]. This general model can be used for easily-partitionable or "embarassingly parallel" problems. I have used this model to implement a number of applications including factoring [27], distributed web-crawling [27], and fractal rendering (i.e., Mandelbrot set) [27], and have also supervised students who have written a code breaking application for the RSA RC5 challenge [27], and a general sub-framework for parallel genetic algorithms [28]. Recently, I have also used the master-worker model to implement the *Bulk Synchronous Parallel* (BSP) programming model [27]. This model [27] makes programming easier by providing programmers with familiar remote memory access and message-passing functions, while enforcing timing semantics on these communication operations that make implementation in a volunteer computing environment relatively easy. By implementing BSP with Bayanihan, we hope to enable programmers to port the growing collection of BSP programs [26] to Java and make it easier to do realistic research using volunteer computing systems.

- *User Interface Design*. The framework's modular design gives programmers great flexibility in designing user interfaces. For example, programmers can create

application-specific GUI objects to view the same object in different ways, as well as *generic GUI objects* to view different objects in the same way. The framework also enable programmers to write *watcher applets* that allow users to view results and control computation interactively using web browsers.

- *Adaptive Parallelism.* The master-worker model employs a form of *eager scheduling* [26], a simple technique for adaptive parallelism wherein fast workers which run out of new work to do are allowed to bypass slow or crashed workers and take over their work load when necessary [27]. Eager scheduling allows the computation to take full advantage of available computational power while providing resiliency in the face of volunteers leaving or crashing at anytime.

- *Fault and Sabotage Tolerance.* To protect against both unintentional faults and malicious sabotage, I have implemented transparent support for traditional replication and voting techniques, as well as *spot-checking*, wherein the server randomly serves workers jobs with known results, *blacklists* those that return wrong results, and *backtracks* through all other results depending on them. Early experiments produced promising results. Spot-checking, for example, even without blacklisting, was significantly more efficient than voting and was able to maintain an error rate of only a few percent even in cases where there were *more* saboteurs than honest volunteers [27].

- *Performance and Scalability.* Sequential timing tests show that with *just-in-time* (JIT) compilation, sequential Java performance is comparable to that of native C code for highly repetitive computational code [27,27]. Parallel tests with up to 16 workers produced good speedups (i.e., about 90% of the ideal) for coarse-grain computations. However, as in all NOW-based parallel systems today, performance degrades as the problem granularity decreases [27]. To reduce congestion in large-scale systems or systems with slow network links, I have also implemented basic mechanisms for *volunteer servers* that can be used by moving communication load away from the central server [27].

REMAINING WORK

In the next few months, I intend to wrap up this thesis research, and write the thesis document itself, which I hope to finish and defend by April 1999. While a lot of research work has already been done, there are still some loose ends that I hope to tie before I finish. These include:

- *Cleaning-up the framework for public use.* Experiences with students using my framework has shown that while the framework is usable and flexible, it is poorly documented, and requires a steep learning curve. I plan to clean up the code to make it easier for others to understand and use.

- *Improving Bayanihan BSP performance.* The recently-developed Bayanihan BSP programming model shows great potential for making it easy to write a wide variety of practical volunteer computing applications. The current *implementation* of this model, however, is only a proof-of-concept implementation that sacrifices performance and scalability for easy implementability. To be useful for real applications, we need to improve this implementation by adding mechanisms such as lazy checkpointing and migration, caching, and peer-to-peer communication [27].

- *More fault and sabotage tolerance.* Although the fault-tolerance results presented in [27] are very encouraging, they are clearly just the beginning of much potential research. One interesting open question is what happens with spot-checking when saboteurs do not *always* give wrong answers, or when they give a wrong answer once and then stop. It seems that while doing so can reduce the chances of a saboteur getting caught and thus increase error rate, it also reduces the rate at which a saboteur submits erroneous results. Thus, intuitively, we might expect that there is a limit to maximum *absolute* error rate that a saboteur can cause no matter what strategy it uses. Another interesting research possibility is that of finding ways to combine voting, spot-checking, backtracking, and blacklisting to reduce error rates even further. I hope to make progress towards answering these questions through a combination of theoretical mathematical analysis and empirical testing. For the latter, I plan to develop a simulator that will allow us to try the fault-tolerance mechanisms with large numbers of (simulated) volunteer workers, some of whom can be saboteurs. If there is still time, I may also try to supplement the existing fault-tolerance techniques with digital signatures, checksums, dynamic obfuscation, to reduce the error rate even further. However, I think this is unlikely.

- *More practical applications.* I hope to demonstrate Bayanihan's versatility and practical value by using it in more practical and realistic applications. The BSP model, if implemented efficiently, can make this task dramatically easier, as it will allow programmers to port existing BSP programs as well as write new applications using a familiar model. I plan to give my framework to researchers in Japan and the Philippines, and help them use it for their own applications. As new applications for Bayanihan are developed, I plan to collect and study timing results from these applications.

In the end, I plan to produce a simple but usable prototype software package fit for public use that would allow a programmer to set up a simple volunteer computing system within his or her own intranet.

CONTEXT

Project Bayanihan contributes a unique approach to a growing number of other projects developing general-purpose software packages for metacomputing in general and Java-based volunteer computing in particular.

PREVIOUS AND RELATED WORK

Early metacomputing systems include Condor [12], started in 1988 from the University of Wisconsin in Madison, which allowed programmers to harness the idle cycles of existing computers for computational use. Others allowing the use of networks of workstations to provide inexpensive supercomputing facilitites include the NOW project at U.C. Berkeley [3] (from which the acronym originated), and the Parallel Virtual Machine (PVM) library developed at Oak Ridge National Laboratory [14], which now enjoys widespread use among universities and research labs for both research and teaching purposes.

More recent metacomputing projects have started going beyond university-wide networks, and addressing the problems of security, reliability, and scalability in nation-wide and global networks. These include Legion [23] at the University of Virginia, and Globus [17] at the Argonne National Laboratory and the University of Southern California. Perhaps the biggest project so far is the Partnerships for Advanced Computational Infrastructure (PACI) program, which will receive $340 million from the National Science Foundation over the next five years to develop a "National Technology Grid" for the United States [36]. Unfortunately, although these systems are very flexible and very powerful, their complexity set up requirements currently prevents them from being used outside of research labs with the necessary expertise.

In the past two years, several Java-based metacomputing projects have been started that seek to drastically reduce setup requirements and bring metacomputing closer to the reach of the common user. Most notable of these are probably Charlotte at New York University [4], which provides the programmer with an easy-to-use framework for developing applications using the "eager scheduling" form of adaptive parallelism, and SuperWeb/Javelin at U.C. Santa Barbara [2], which originally proposed the idea of Java-based commercial market systems, and is actively seeking to develop the mechanisms for it. Although these projects do not use the term "volunteer computing" (which I coined), their ideas and goals are basically the same as those I have discussed here. (In fact, although most of the basic ideas presented here were developed independently, their present form has benefitted from discussions with these groups.)

UNIQUE CONTRIBUTIONS

While the ideas and goals behind Project Bayanihan may seem very similar to those of these other projects, Project Bayanihan brings with it a unique approach. Among the things that distinguish Project Bayanihan from other projects are:

1. *Focus on Accessibility.* Unlike large projects such as Legion, Globus, and PACI, Project Bayanihan balances the need for complex features with the need for simplicity and usability by the common user. It is interesting to note, for example, that the largest world-wide computational network built so far, `distributed.net`, did not come from any of these groups, but from a small group of hobbyists who developed a simple way to make it easier for ordinary Internet users to join their network.

2. *Focus on Flexibility.* While current Java-based systems such as Charlotte and SuperWeb provide the programmer with a way to easily generate different kinds of applications, they generally restrict programmers to following a particular programming model. With Project Bayanihan, we are making full use of object-oriented techniques to build a general-purpose framework for implementing and experimenting not only with different applications, but with different programming models and mechanisms as well.

CONCLUSION

In this electronic age, it may be hard to find people still carrying their neighbors' houses on long bamboo poles – even in the villages of the Philippines. But in the global village that the Internet has become, the community spirit of *bayanihan* can live on through volunteer computing. With my thesis project, Project Bayanihan, I hope to develop theory and technology that, much like the bamboo poles of old, will serve as the underlying framework that will enable people around the world to work together as one and accomplish the seemingly impossible.

REFERENCES

1. Harold Abelson, Thomas F. Knight, Gerald Jay Sussman, et al. Amorphous Computing. http://www.swiss.ai.mit.edu/~switz/amorphous/
2. A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman. SuperWeb: Research Issues in Java-based Global Computing. *Concurrency: Practice and Experience.* Vol 9(6). June 1997. pp. 535-553.
3. Thomas E. Anderson, David E. Culler, David A. Patterson, and the NOW Team. A Case for Networks of Workstations: NOW. *IEEE Micro*. Feb, 1995. http://now.cs.berkeley.edu/Case/case.html

4. Arash Baratloo, Mehmet Karaul, Zvi Kedem, and Peter Wyckoff. Charlotte: Metacomputing on the Web. In Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems, Sept. 1996. http://cs.nyu.edu/milan/charlotte/
5. Adam L. Beberg, Jeff Lawson, and David McNett. distributed.net. http://www.distributed.net/
6. Blumofe, et al. Cilk: An Efficient Multithreaded Runtime System. In *Proceedings of the 5th ACM SIGPLAN Symposium on Principles of Parallel Programming* (PPOPP'95), Santa Barbara California, July 19-21,1995. pp. 207-216. http://theory.lcs.mit.edu/~cilk/
7. BSP Worldwide Home Page. http://www.bsp-worldwide.org/
8. P. Cappello, B. O. Christiansen, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-Based Parallel Computing Using Java. *ACM Workshop on Java for Science and Engineering Computation*. Las Vegas, June 1997. http://www.cs.ucsb.edu/research/superweb/
9. Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*. 32(4):444-458, April 1989.
10. Digital Equipment Corporation. Digital's Altavista Search Site Unveils Largest and Freshest Web Index. Press Release. Maynard, Mass. Oct. 14, 1997. http://www.altavista.digital.com/av/content/pr101497.htm
11. `distributed.net`. Official List Announcement. Oct. 22, 1997. http://www.distributed.net/pressroom/56-announce.html
12. Scott Fields. Hunting for Wasted Computing Power: New Software for Computing Networks Puts Idle PC's to Work. Research Sampler. University of Wisconsin-Madison. 1993. http://www.cs.wisc.edu/condor/doc/WiscIdea.html
13. Gamelan home page. http://www.gamelan.com/
14. Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallelism*. MIT Press, 1994. http://www.netlib.org/pvm3/book/pvm-book.html
15. David Gelernter and David Kaminsky. Supercomputing out of recycled garbage: Preliminary experience with Piranha. In *Proceedings of the 1992 ACM International Conference of Supercomputing*, Washington, D.C., July 1992.
16. W. Wayt Gibbs. CyberView. *Scientific American*, May 1997. http://www.sciam.com/0597issue/0597cyber.html
17. The Globus Project. http://www.globus.org/
18. Junzo Hagimoto. Construct Java applications through distributed object technology. *JavaWorld*. Dec. 1997. http://www.javaworld.com/javaworld/jw-12-1997/jw-12-horb.html
19. Hirano Satoshi. HORB: Distributed Execution of Java Programs. *Worldwide Computing and its Applications*. Springer Lecture Notes in Computer Science 1274, 1997. pp. 29-42. http://ring.etl.go.jp/openlab/horb/

20. John H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press. 1992.
21. M. Hostetter, D. Kranz, C. Seed, C. Terman, and S. Ward. Curl: A Gentle Slope Language for the Web. MIT Laboratory for Computer Science. White Paper. http://www.cag.lcs.mit.edu/curl/wwwpaper.html
22. James E. Katz and Philip Aspden. A Nation of Strangers? *Communications of the ACM*. Vol. 40(12). Dec. 1997. pp. 81-86.
23. Legion: Worldwide Virtual Computer. University of Virginia. http://www.cs.virginia.edu/~legion/
24. Steven Levy. Wisecrackers. *Wired*, Issue 4.03, March 1996. http://www.hotwired.com/wired/4.03/features/crackers.html
25. Media One home page. http://www.mediaone.com/
26. NASA. Pathfinder Gets Hit Hard on the Internet. Internet Advisory, July 9, 1997. ftp://ftp.hq.nasa.gov/pub/pao/internet/1997/i97-8.txt
27. NCSA. The Metacomputer: One from Many. http://www.ncsa.uiuc.edu/Cyberia/MetaComp/MetaHome.html
28. Michael Rappa. Solomon's House in the 21st century. Working Paper. Nov., 1994. http://web.mit.edu/technika/www/solomon.html
29. Lydia Sandon. Advanced Undergraduate Project. MIT, Spring 1998.
30. Luis F. G. Sarmenta. Bayanihan: Web-Based Volunteer Computing Using Java. In *Proceedings of 2nd International Conference on World-Wide Computing and its Applications* (WWCA'98), Tsukuba, Japan, March 1998. Published as Springer Lecture Notes in Computer Science 1368, 1998. pp. 444-461. http://www.cag.lcs.mit.edu/bayanihan/papers/wwca98/html/
31. Luis F. G. Sarmenta, Satoshi Hirano, and Stephen A. Ward. Towards Bayanihan: Building an Extensible Framework for Volunteer Computing Using Java. In *Proceedings of ACM 1998 Workshop on Java for High-Performance Network Computing*. Palo Alto, Feb. 1997. Published as Concurrency: Practice and Experience., Vol. 10(1), (1998). http://www.cag.lcs.mit.edu/bayanihan/papers/acm98p/html/
32. Luis F. G. Sarmenta, Satoshi Hirano. Building and Studying Web-Based Volunteer Computing Systems Using Java. *Future Generation Computer Systems*, Special Issue on Metacomputing (to be published 1999). http://www.cag.lcs.mit.edu/bayanihan/papers/fgcs/html/
33. Luis F. G. Sarmenta. An Adaptive, Fault-tolerant Implementation of BSP for Java-based Volunteer Computing Systems. Submitted to IPPS'99 International Workshop on Java for Parallel and Distributed Computing, Puerto Rico, April 1999. http://www.cag.lcs.mit.edu/bayanihan/jpdc/jpdc.ps
34. SETI@home home page. http://www.bigscience.com/setiathome.html
35. D. Skillicorn, J.M.D. Hill, W.F. McColl. Questions and answers about BSP. *Scientific Programming*. 6(3) (1997) pp. 249-274.
36. Larry Smarr. Toward the 21st Century. Communications of the ACM. Vol. 40(11). Nov.1997. pp. 29-32.
37. Sun Microsystems. Java home page. http://java.sun.com/

38. Eamon Walsh. Design and Implementation of a Framework for Performing Genetic Computation Through a Volunteer Computing System. Research Science Institute summer mentorship program report. MIT, August 1998.
39. George Waltman. Mersenne.org Main Page. http://www.mersenne.org/
40. WebTV home page. http://www.webtv.com/ns/index.html
41. William Wulf. The Collaboratory Opportunity. *Science*, Aug. 1993.