

Frontier Journal

Frontier Visionary Interview

[Prof. Robert J. Aumann](#), Hebrew Univ. of Jerusalem, Nobel Laureate in Economics
[Prof. A. Michael Spence](#), Stanford Univ., Nobel Laureate in Economics
[Prof. Martin L. Perl](#), Stanford Univ., Nobel Laureate in Physics
[Prof. Frank Wilczek](#), MIT, Nobel Laureate in Physics
[Steve Wozniak](#), Co-founder, Apple Computer
[Vinton G Cerf](#), Turing Award Winner
[Ann Winblad](#), Co-founder, Hummer Winblad Venture Partners
[Richard Stallman](#), Founder of GNU Project
[Jim Rogers](#), American Investor
[Alan Kay](#), PhD, Turing Award Winner
[Prof. Bjarne Stroustrup](#), Man behind C++, Texas A & M Univ.
[Brian Behlendorf](#), Co-founder of Apache Project
[Rajeev Madhavan](#), Co-founder, Chairman & CEO, Magma Design Automation
Jimmy Wales, Founder of Wikipedia
[Craig Newmark](#), Founder of Craigslist.org
[Greg Gianforte](#), Founder & CEO of RightNow Technologies, Inc
[Grady Booch](#), Chief Scientist, IBM Rational
[Aart de Geus](#), PhD, Co-founder, Chairman & CEO, Synopsys

Content

Free Factories: Unified Infrastructure for Data Intensive Web Services
Alexander Wait Zaranek et al.4

Augmenting RAID with an SSD for Energy Relief
Hyo J. Lee et al.32

Energy Aware Consolidation for Cloud Computing
Shekhar Srikantaiah et al.43

Sonic Nirvana: Using MEMS Accelerometers as Acoustic Pickups in
Musical Instruments *Rob O' Reilly et al.*54

Analog Front End for 3G Femto Base Stations Bring Wireless Connectivity
Home

Free Factories: Unified Infrastructure for Data Intensive Web Services

Alexander Wait Zaranek, Tom Clegg, Ward Vandewege, George M. Church

Harvard University

await@genetics.med.harvard.edu

Abstract

We introduce the Free Factory, a platform for deploying data-intensive web services using small clusters of commodity hardware and free software. Independently administered virtual machines called Freegols give application developers the flexibility of a general purpose web server, along with access to distributed batch

processing, cache and storage services. Each cluster exploits idle RAM and disk space for cache, and reserves disks in each node for high bandwidth storage. The batch processing service uses a variation of the MapReduce model. Virtualization allows every CPU in the cluster to participate in batch jobs. Each 48-node cluster can achieve 4-8 gigabytes per second of disk I/O. Our intent is to use multiple clusters to process hundreds of simultaneous requests on multi-hundred terabyte data sets. Currently, our applications achieve 1 gigabyte per second of I/O with 123 disks by scheduling batch jobs on two clusters, one of which is located in a remote data center.

1 Introduction

We built Free Factories to help the PGx team win the Archon X PRIZE for Genomics and to meet the needs of the Personal Genome Project. The prize is awarded for sequencing one hundred complete human genomes in less than ten days [29]. Doing this with Polony sequencing [17,25] and related technologies [30,31], as the PGx team plans to do, will involve distilling many petabytes of raw data to produce about 100 gigabytes of output. This DNA sequencing capacity can be used to help build a database of personal genome-phenome data sets; coupled with a data mining and analysis engine, this will provide opportunities for many new discoveries.

Storing and analyzing data at this scale still requires exotic computing systems [3,15,28] - many scientists, physicians and members of the general public would like to participate in the development of these technologies, but do not have access to the resources they need to get started. Furthermore, anyone creating a database of sensitive personal information has to address privacy and disclosure concerns, and there is no single correct way to do that. The Personal Genome Project aims to overcome these obstacles and, ultimately, give individuals the tools to make genetic discoveries of their own [7,19,20].

Figure 1: A Free Factory contains about five clusters of 12 to 48 nodes. Some clusters are colocated with data acquisition instruments; their sizes are limited by the available power, cooling, and space. Other clusters are

located in data centers. The clusters are interconnected by relatively slow networks.

`\includegraphics[width=2.5in]{0_data_personal_ward_Desktop_freelogy_svn_lyx_2008_usenix_figs_ff.eps}`

To help alleviate these barriers, we consider the needs of a relatively small organization that supports several independent applications. Some such organizations support scientists in research activities, like developing new sequencing technologies; some focus on medical applications, like predicting which treatments will be effective for a particular patient. Within each of these communities, scientists and application developers generally benefit by sharing data and computing resources, although they may need to segregate some data and resources in order to satisfy a particular security model. End users choose who to trust with their personal data, establish legal agreements with those trustees to control how their data is used, and access relevant information using web services.

With the Free Factories platform, we make the first steps toward this vision. We emphasize efficiency in a range of installation sizes, from a single 12-node cluster to several 48-node clusters. We cater to data-intensive applications that are conducive to parallel computation, but are limited by the ability of storage systems to support many concurrent tasks. We avoid proprietary software and expensive hardware. Organizations can start working with substantial data sets on small clusters, and expand their capacity by adding new clusters. Multiple small clusters also provide opportunities to enforce different privacy and data integrity guarantees for different applications and data.

Small installations of low-cost hardware provide processing and storage capacity at the scale we need for these applications, but efficient and fault-tolerant utilization of this hardware is non-trivial. We have used a pragmatic approach of selecting and arranging free software to make the best of the performance characteristics of cheap hardware. Our goal is to utilize 90% of our hardware capacity, including disk I/O bandwidth, network bandwidth, and CPU time. The result is a unified platform that makes efficient use of hardware in an environment where a

variety of users and applications share storage and computation resources. We encourage others to deploy and develop this platform further. [16]

2 Design and architecture

A Free Factory provides hosting, data storage, and batch processing services for a number of web applications. These applications involve data-intensive computation: they are conducive to asynchronous parallel processing, but their performance is limited by the available disk I/O bandwidth. Their demands for CPU time are highly variable, so it is sensible for them to share a pool of CPU resources by submitting batch jobs. They also tend to share data sets with one another, so it is sensible to share a large data storage system. The application developers have common goals rather than being in competition, so it is beneficial to let them see the source code and results of one another's batch processing jobs. The applications themselves may be maintained by different development teams, so each application should run in its own independent virtual machine.

We identify the following roles in the Free Factory environment. "Users" – scientists, physicians, and members of the general public – are interested in a web service and interact with it via a web browser. "Administrators" maintain the Free Factory infrastructure. A "trustee" sets policy and obtains funds to pay for staff, hardware, and hosting. "Developers" are the application developers and scientists who maintain Freegols. "Freegols" are web services that utilize cluster computing and storage resources. The term Freegol, or Free Golem, emphasizes the idea that the web services are developed and maintained independently of the cluster infrastructure, and independently of one another.

The canonical Free Factory (Figure 1) contains about five clusters with 12 to 48 nodes each. Some clusters can be co-located with data acquisition engines such as DNA sequencing instruments. Each cluster acts as a web hosting platform for several applications, as well as supporting the data storage and batch processing needs of those applications. A Free Factory of this size can be maintained by three administrators.

Each cluster is constructed using 1U rack-mount machines with big disks and inexpensive CPUs. Today, each of these low-cost machines offers about 240 MB/s of disk I/O bandwidth as well as 2 Gb/s of network bandwidth. With data and processing resources striped across an entire 48-node cluster with 192 disks, it is theoretically possible to achieve 11 GB/s of disk I/O during a batch job. The two clusters we have built contain 85 and 38 disks respectively. At 60 MB/s per disk this gives us 5.1 GB/s and 2.2 GB/s of available disk bandwidth respectively.

We use virtualization to deploy cache, storage, and batch processing services on every single node in each cluster: CPU-intensive jobs can make use of every CPU in the cluster, while data-intensive jobs can make use of every disk. This layout allows us to achieve high I/O throughput even while many concurrent processes are working on the same data set. We have achieved as much as 1 GB/s of I/O on a cluster with many concurrent processes; this compares favorably with a 12-disk RAID-6 system, which we have to limit to a single reader in order to achieve a sustained throughput of 100 MB/s.

Figure 2: Dotted lines denote virtual machines. Batch processing workers in warehouse instances are dispatched by the batch controller on behalf of Freegols. RAM cache, disk cache, and long term storage services are accessed by Freegols and batch jobs using the same client library.

`\includegraphics{1_data_personal_ward_Desktop_freelogy_svn_lyx_2008_usenix_figs_components-and-vms.eps}`

On each machine, a "warehouse instance" runs the processes that implement cache, storage, and batch processing services. Warehouse instances are implemented as virtual machines on the nodes that are used for hosting Freegols, and consume entire physical machines in other cases. Each cluster manages its own cache, storage, and batch processing services using a number of controller processes that run in a virtual machine. Freegols and batch programs use the warehouse client library to communicate with these controller processes and the warehouse

instances' service processes (Figure 2).

2.1 Commodity hardware, free software

When building an affordable, high-availability, data-intensive web service it is desirable to have control of the system's total cost of ownership. Part of our strategy is to avoid proprietary technologies in favor of free software. This way, we can build on existing tools and have confidence that we can replace or modify them when necessary.

When choosing hardware, we are interested in maximizing the usable disk, RAM, CPU, and network bandwidth per unit cost. At present dual gigabit ethernet, one terabyte SATA disks, and dual-socket quad core motherboards seem to best fit our needs. (We prefer larger disks even at a higher cost per gigabyte because disks have high failure rates independent of size [2,12,23,24] and manual intervention is expensive even in a fault-tolerant system.) Full-bandwidth 48-port switches are also available at low cost. Therefore, the most affordable way to configure a large number of disks and CPUs today is to build a number of 48-node clusters, interconnected by relatively slow network links or virtual private networks.

Given the limited size of each cluster, scalability requires that applications have access to more than just one cluster. We expose the network topology to the applications so that they can make informed decisions about where and when to perform computation, and where to store data, depending on the varying availability of these resources on different clusters. (Our intent is for future client libraries to help applications make good scheduling decisions; currently it is practical for a developer to select one of our two clusters when running a job.)

Each cluster is self-contained; hardware and network failures do not cascade to other clusters. The small cluster size makes it feasible to deploy entire clusters at once, rather than performing incremental upgrades to a large cluster. Multiple clusters can be used to increase confidence in the repeatability of computational results, and to

monitor the effects of different combinations of hardware, software, and usage patterns.

To illustrate the cost of a Free Factory we consider purchasing a 48-node cluster with 192 1 TB disks for \$170,000. Annual operating costs include \$27,000 of power (18 kW at \$0.17/kWh), \$25,000 for network access and floor space (at the rate we pay at Harvard), and \$50,000 for a part-time administrator. Thus, the total cost to deploy the cluster is \$272,000 for the first year, and \$102,000 per year thereafter.

For a point of reference, we consider Amazon Web Services, a popular computing platform that allows an organization to pay for computation and storage on an as-needed basis. This is often less expensive than using dedicated hardware because the cost of processing is determined by average demand rather than peak demand. However, for the data-intensive applications we consider here, the strategy of frequently allocating and releasing compute nodes is less beneficial because of the time spent copying data to and from the nodes each time. In effect, a CPU-on-demand system requires a higher allocation rate in order to do the same work, compared to a dedicated hardware approach where data is kept close to the processors and can be read at full speed whenever it is needed.

We overlook this distinction for the sake of making a direct comparison with the Amazon EC2 and S3 services [1]. Amazon EC2 offers an "extra large instance" with two 1 TB disks and four virtual CPUs for \$0.80 per hour. Thus, a 48-node cluster is roughly equivalent to 96 extra large instances. If the cluster achieves 25% CPU utilization, its value is comparable to 24 extra large instances at \$168,000 per year. Meanwhile, S3 provides long term storage for \$0.15/GB. At this rate, it costs an additional \$43,000 per year to store 24 TB of data (half of the long term storage capacity of the cluster). The actual amount of data transferred to and from S3 depends on the application; if 15 TB is transferred to Amazon at \$0.10/GB, and 2 TB is transferred out of Amazon at \$0.17/GB, then the transfer cost is \$1,840 (traffic between EC2 and S3 is free). The total cost of the Amazon service over two years is \$424,000, compared to \$374,000 for the first two years of a Free Factory.

2.2 Freegols and virtualization

There are a wide variety of languages, toolkits and methodologies for deploying scalable web services. One factor that contributes to Amazon EC2's popularity is that it permits web service developers to choose their own tools. We found that giving developers this freedom suited our environment too.

Virtualization encourages a model wherein developers have "root" privileges on their own virtual servers, or Freegols. Typically, a Freegol is configured as an Ubuntu server with common application server software like Apache and MySQL. The warehouse client library is easy to install and upgrade using the native package manager. Our goal is to make it easy for developers to start using Freegols to deploy services; part of this strategy will be to port the Perl client library to other popular languages like Python.

RAM, virtual processors, and network bandwidth are shared among Freegols, cache and storage service processes, and batch jobs. If necessary, a developer can ensure that a Freegol does not share these resources with other Freegols by getting an allocation for all of the available CPU and RAM.

In addition to Freegols, it is often beneficial to set up virtual machines on the cluster for applications that do not use the cache, storage, or processing services. Cluster administrators are likely to use virtual machines to deploy common network services like web proxies, DNS caches, and backup servers.

2.3 Cache and storage services

The objectives of our storage services are to: (1) minimize I/O bottlenecks in order to make the best use of available CPU cycles; (2) provide a low-latency shared cache with automatic garbage collection; (3) provide long term storage with high read and write throughput and provisions for usage accounting. The storage services must yield good performance when used directly from Freegols, as well as from batch jobs. Inspired by the Google

File System [13], Bigtable [6] and the plethora of raw materials made possible by free and open source software, we felt we could build a system that suited our needs perfectly.

We implement a three-level content addressable storage service. We use Memcached [9] as a low latency RAM cache. This is well suited to small strings (less than one megabyte) and it works well even with many concurrent clients because it does not employ a central controller. We use MogileFS [9] to implement a cluster-wide distributed disk cache. This gives good performance for block sizes up to 64 MiB. For long term storage we have developed software that minimizes the role of a central controller while providing opportunities for usage accounting. All of these storage and cache services are accessible to all applications and batch jobs in the Free Factory.

Aggregate I/O bandwidth is limited by several factors. Our storage services are designed to minimize the effects of these factors.

1. Disk seeks reduce aggregate disk read and write bandwidth. We minimize seeks by storing data in contiguous 64 MiB segments when possible, and ensuring that each segment read/write operation is not interrupted by any other disk activity. This means that readers tend to wait longer before they start to receive data, which is why general purpose operating systems do not use this strategy; however, in this environment, high throughput is more valuable than low latency.

2. A gigabit network interface can only handle two concurrent readers at full disk transfer rates. To prevent this from limiting throughput when many concurrent processes are accessing the same data set, we stripe every data set across all of the cluster nodes.

3. Central controllers get bogged down when they try to handle too many concurrent clients. Our storage service can read and write blocks without involving the storage controller in real time. Our disk cache is indexed in RAM, so most reads do not involve the controller.

4. Writes are slow because robustness requires storing multiple copies of each block. When an identical copy of an output block already exists in the storage service, our content-addressing approach allows the client library to transparently skip unnecessary write operations.

The cache and storage services use an MD5 addressing approach: the name of each block of data is the MD5 checksum of the data. This naming scheme provides several benefits.

1. The client library, when retrieving a block from the cache and storage services, computes the MD5 checksum of the data and compares it with the block name. If the checksum does not match, the client library can try to fetch the data from a different host, a different storage service, or a remote cluster. This verification process is completely transparent to the application.

2. Multiple jobs often produce the same output. For example, a Freegol may re-run a job every time the job's source code is modified in the source repository, and every time an operating system upgrade is performed, to make sure the job still produces the same result. If the output is identical to the previous run, no additional storage space is consumed by the new job. We encourage developers to make use of this property by segregating their main output from their diagnostic messages (which are likely to contain timestamps and the like, but are usually small) and by avoiding non-deterministic outputs (sometimes this involves simple tricks like using the "Minimal" option in the IO::Compress::Gzip Perl module).

3. Freegols and concurrent batch jobs can share data without worrying about overwriting blocks at inopportune times.

Our simple tests have demonstrated that reading 64 MiB files sequentially results in throughput exceeding 90% of a disk's maximum sustained transfer rate. Real data used by Freegols, on the other hand, is likely to include many smaller files. To help Freegols achieve high throughput when working with smaller files, we introduce a "manifest" file format. In addition to increasing performance, the manifest format is a valuable tool for managing

large data sets.

A manifest is an index to a collection of data files, analogous to a directory tree in a traditional filesystem. It is stored as a plain text file. Each line of the text file represents a "stream"; each stream contains a set of data files. The content of the data files is stored in a manner similar to a UNIX tar archive: the data from all files in a stream are concatenated, the result is split into 64 MiB blocks, and the blocks are written to cache or storage. The manifest file specifies the MD5 checksums and sizes of these data blocks, as well as the names of the individual files and their positions within the stream. The manifest file itself can be split into 64 MiB blocks and stored, and its unique key – the list of MD5 checksums of those blocks – can be used to retrieve it. (If this list of checksums is inconveniently long, the list itself can be stored in a separate block, whose MD5 checksum then serves as a more concise key to the large manifest.)

This manifest format has several noteworthy features. It is concise: a short key is enough to specify a large collection of data. It is portable: if two jobs running on different clusters produce identical output, the resulting manifest keys are also identical. The integrity of the data blocks, and the manifest itself, are easily verified. It is efficient to read an entire stream worth of data from disk, even if the stream represents many small files. However, random access – reading and writing small files in various streams out of order – is not efficient. We expect applications to be cognizant of this restriction, and read and write entire streams whenever possible.

Once a manifest is written to the cache or storage service, it can be retrieved, or used as the input to a batch job, by any Freegol that knows its key. Also, each cluster has a central database of manifest names. To attach a name to a manifest, a Freegol sends a request to the cluster's storage controller specifying the manifest key, the desired name, and – to avoid race conditions – the manifest key that was previously associated with the name. Naming a manifest has several consequences.

1. Any Freegol can look up the name to retrieve the manifest key.
2. The data set is considered to be valuable to the signer. If the blocks referenced by the manifest are in long term storage, those blocks should not be deleted.
3. The old manifest, if it is not associated with any other names, is no longer considered valuable; the data blocks mentioned in it may be deleted if they are not mentioned in any other named manifests.

Optionally, a Freegol may also specify a list of PGP keys indicating entities that have permission to overwrite this manifest name.

One drawback to content-addressable storage is that in-place updates are inefficient. For example, if a 32 MiB stream is written, and a new version of the stream is written afterward that has 16 MiB of additional data appended to the original 32 MiB, then both the 32 MiB version and the 48 MiB version may be written to disk. We find this acceptable for the following reasons. The manifest format allows the 48 MiB stream to be stored by referencing the original 32 MiB block followed by the new 16 MiB block, if the existence of the 32 MiB block is known when the second version is written. In any case, we are willing to sacrifice some storage space in order to avoid race conditions.

This illustrates one of the useful aspects of the manifest format. A manifest key specifies the data itself, not just a set of filenames. If a modified version of a large data set appears, the previous key can still be used to access the old data, and a program requesting the original version will never unexpectedly receive the newer data. This simplifies application design, and provides a significant practical benefit for scientific applications and other environments where repeatability is of major concern.

Periodic garbage collection is inexpensive. The storage controller can quickly read all of the manifests that appear in the manifest name database, and produce a list of blocks that are still in use. The 64 MiB block size ensures

that the resulting list is small compared to the amount of stored data. Garbage collection is accomplished by comparing this list against the list of blocks stored on disk. We have not yet implemented automatic garbage collection but we have found that a list of 588,000 distinct block names, representing 11 TB of data referenced by 1151 distinct manifests, can be generated in 70 seconds.

2.4 Batch processing services

The objectives of our batch processing services are to: (1) use as many as possible of the available CPU cycles on all machines; (2) make it easy to repeat jobs many times on various clusters to check for bugs and inconsistencies; (3) handle occasional failures gracefully; (4) keep statistics about performance and failure rates.

Batch processing is coordinated by a batch controller on each cluster. The batch controller accepts requests from Freegols to schedule new jobs. The batch controller starts new jobs when the requested number of warehouse instances become available. Freegols can expect the batch controller to occasionally pause and resume a job, or reduce its resource allocation, depending on subsequent job submissions. (Our current implementation uses a simple greedy scheduling algorithm, and the batch controller only pauses and resumes jobs when specifically requested by a Freegol.)

Freegols can retrieve a list of current, pending, and previous jobs from the batch controller. This list includes specifications and statistics for each job, including inputs, outputs, start and finish times, and (for active jobs) what portion of the job has been completed so far. Freegols can poll the batch controller to determine the status of their own jobs, get hints about how busy the cluster is, and look up details of jobs that other Freegols have submitted.

The execution of a batch job is supervised by a job manager process running on the same virtual machine as the batch controller. The job manager supports a computation strategy similar to MapReduce [11]. Each job consists

of a number of steps, each of which is performed on a single warehouse instance. Each job step stores some output in the cache; the job manager assembles the output into a manifest at the end of the job. Additionally, each job step has the ability to enqueue more job steps.

The program that performs the work of a single job step is called a "mr-function" (from "MapReduce function"). Mr-functions are kept in a revision control system. Administrators and developers can update existing mr-functions and create new ones, subject to access controls on the revision control repository. Once it is committed to the repository, a mr-function can be used in a job submission by any Freegol.

The most convenient way to construct a batch job is to use a single manifest as input, and schedule one job step for each stream in the manifest. Each job step reads one full stream from the input manifest from start to finish, and writes one full stream in the output manifest. The client library comes with tools and examples to make it easy to write mr-functions that use this strategy.

3 Implementation

3.1 Commodity hardware, free software

We are currently operating two clusters using a variety of commodity off-the-shelf hardware, free software such as GNU/Linux [14] and our own custom software that is released under the GNU GPL [16]. Our two clusters are located a few kilometers apart and connected by Harvard University's fiber optic network. "Uncle" is our experimental research cluster; "templeton" is our production cluster. Our clusters are depicted in Figure 3. Uncle is largely made up of 32-bit dual-CPU Intel Xeon machines, many of which are four years old. Templeton's hardware is more recent: each machine has two dual-core AMD Opteron 64-bit processors.

Figure 3: On each cluster, a few physical machines run the Xen hypervisor (mixed dark/light gray blocks). These machines are partitioned into warehouse instances, Freegols, and other virtual machines. Other physical machines

are dedicated to warehouse instances (dark gray only).

`\includegraphics[width=3in]{2_data_personal_ward_Desktop_freelogy_svn_lyx_2008_usenix_figs_networkdiagram.eps}`

Each cluster consists of 47 machines. All newer machines have four disk slots and the older machines are diskless. Each newer machine has two gigabit ethernet ports, which are connected to two 48-port gigabit switches. Two to four "headnodes" have a third ethernet port connected to the upstream switch, and optionally a fourth port connected to an out of band management network. The headnodes act as gateways to the internet and as VPN endpoints.

Our VPN is a simple OpenVPN [22] point-to-point setup, terminated on a headnode at each end. The VPN data rate reaches about 200 Mb/s. Throughput is limited by the processing speed of one endpoint that has one dual-core Opteron 265 CPU at 1.8 GHz. The other endpoint has 2 single-core Opteron 250 processors at 2.4 GHz; CPU load is considerably lower there.

Uncle currently runs the latest release of Ubuntu [5], a popular flavor of the Debian [27] GNU/Linux system that many lab members run on their desktops, while Templeton runs the latest "long term support" release of Ubuntu. We chose Ubuntu over Debian because of Ubuntu's predictable six month release schedule, but we expect to build a Debian cluster in the future. We like the Debian and Ubuntu philosophy and have found that packaging our software using Debian/Ubuntu tools to be a good way to automate the installation of the client library and its dependencies. We believe that inclusion in major community projects, such as Debian, is an excellent way to both reach a wider audience and to further improve our installation automation. Ultimately, we aim to be distribution agnostic.

The latest machines that we purchased came pre-installed with coreboot [8], a free software BIOS that has a number of advantages over proprietary alternatives. All source code is open and available under the GPL. Serial

console support is reliable. Boot time is much faster: coreboot takes only a few seconds to bring the machine into a state where it can start booting the operating system. Also, we can exactly configure the platform to our needs, which allows us to make the platform both more reliable and more secure.

We use Opendgear [21] CM4148 console servers on each cluster for out-of-band access to the serial console of each physical machine. The Opendgear console servers are embedded Linux machines for which the entire source code is available for download. The company provides instructions for modifying the firmware, and for building the firmware from source. We also use networked power distribution units to allow remote power cycling of any device in the cluster via our out-of-band management network.

Aside from the Linux kernel, our software relies on many other open source packages. Notably, Slurm and Munge [18] provide authenticated inter-process communication between the warehouse instances and the batch controller. We rely on Slurm to track which warehouse instances are available for running jobs, and to allow the batch controller to execute batch job steps on the warehouse instances. It does this well, with low latency. It is also a convenient tool for administrative tasks like installing packages and updating configuration files on many instances at a time.

Memcached and MogileFS [9] provide the RAM and disk cache services respectively. MogileFS provides distributed storage with low-latency replication. Used in conjunction with Memcached, it performs well as long as there are not too many concurrent writes.

Perl modules from CPAN are used by the client library, controllers, and service programs for HTTP request handling, data compression, and MD5 hashing. The batch controller uses a MySQL database as a job queue and an archive of past jobs. Subversion provides revision control for the programs that run in the batch processing system, as well as the client library and the service software itself.

3.2 Freegols and virtualization

Some physical machines are configured as warehouse instances, dedicated to providing processing, cache, and storage services. Others are partitioned into virtual machines using the Xen hypervisor – always with one virtual machine configured as a warehouse instance, along with one or more Freegols and other virtual machines controlled by the cluster administrators.

To keep configurations simple, we set aside 4 GiB of RAM on each warehouse instance for use in batch jobs, and allocate the remainder to Memcached processes.

On virtualized machines and dedicated warehouse instances alike, we use RAID-1 to protect all of the local filesystems. We have found that dedicating two entire disks to RAID-1 results in an excess of RAID-protected space. It is wasteful to allocate that space to the storage services, which can already accommodate disk and node failures without RAID-1. Linux allows us to partition the first two disks, assign one partition on each to a software RAID-1 array managed by the Linux Volume Manager, and allocate the remaining space to the MogileFS cache. This is more efficient than whole-disk mirrors, but it still forces us to commit to a partitioning scheme early on. Different permutations of Linux RAID and volume management tools could give us greater flexibility, but we have chosen to avoid the extra complexity that would result. In the future we hope iSCSI will provide more flexible options without creating too much work for administrators.

Figure 4: On a physical machine that is running several virtual machines, a software RAID-1 volume is exported to the Linux logical volume manager, which provides space for local filesystems. The remaining portions of the first two disks are used by the disk cache service. The third and fourth disks are dedicated to the cluster's long term storage service.

`\includegraphics[width=2.5in]{3_data_personal_ward_Desktop_freelogy_svn_lyx_2008_usenix_figs_disks.eps}`

Developers, and some of their users, have access to shell accounts on their Freegols. Our security model is largely perimeter-based at this point, because our users are relatively trustworthy. Specifically, a cluster has one virtual machine with an unprivileged account that is shared by all users. To connect to the SSH port on a Freegol, a user must log in to this shared account using an SSH private key, and specify the name of the Freegol in a remote command string. The `authorized_keys` file in the shared account instructs the SSH server to ignore the client-supplied command and instead run a script that establishes a tunnel to the requested Freegol (the SSH server makes the supplied Freegol name available in an environment variable). Before establishing the tunnel, the script checks a list of permitted combinations of SSH public keys and Freegols. This login procedure is easy to express in a `ProxyCommand` directive in the user's SSH client configuration file.

Virtual machines are used for deploying DNS caches and servers, an SMTP server for routing incoming mail, and a local Ubuntu mirror site. One virtual machine runs a dedicated reverse HTTP proxy. All HTTP and HTTPS traffic to the cluster is forwarded by the headnode to this proxy, which forwards each request to the appropriate Freegol and returns responses to the clients.

Since Harvard routinely receives high volume web traffic, we tested whether our setup can also support a high volume web service. We configured the virtual machine running our reverse proxy server with moderate resources: four shared Opteron 265 cores and 1 GiB RAM. We then launched 2,000,000 requests at the proxy server (with `keepalive` disabled to simulate a worst-case scenario), with 500 simultaneous requests each coming from two physical machines on our LAN. All requests were handled without errors and completed in 657 and 660 seconds on client 1 and 2 respectively. The mean time to complete a request was 0.3 seconds. Both clients completed 90% of requests within 0.2 seconds, and 99% of requests in 4.5 seconds. The slowest request from client 1 was completed in 93 seconds while the slowest request from client 2 was completed in 107 seconds. During this test, a few of us continued to use other Freegols from multiple points on the Internet and found no perceptible

difference in response times. We conclude from this that a Freegol on our cluster can withstand popularity spikes without adversely affecting other Freegols.

The cache, storage, and processing services involve several databases and controller processes, which we run on a single virtual machine. It would also be possible to distribute these processes across several virtual machines; we have not thoroughly explored the performance implications of this choice.

Developers frequently benefit from having separate "development" and "production" Freegols for a given application. The virtualization approach makes it easy for us to deploy these quickly at minimal cost.

3.3 Cache and storage services

The cache and storage services consist of three software layers (RAM cache, disk cache, and long term storage), a manifest name server, and a client library that is used by Freegols and batch jobs.

The client library contains most of the intelligence. It helps applications split and combine data into 64 MiB blocks; it chooses RAM or disk cache for different block sizes according to tunable settings; it constructs hashes when storing data; it avoids writing blocks to cache if they are already present; it stores each block on multiple warehouse instances when writing; and it constructs and parses manifests.

The client library is built on the assumption that all of the underlying storage services are unreliable. It verifies data integrity during retrieval operations. It attempts to retrieve blocks from alternate sources when data is corrupted or missing.

The aim of the storage services is to maintain the highest possible aggregate throughput when reading and writing blocks.

Our RAM cache uses Memcached. The Memcached client library and servers implement a distributed shared-nothing hash table. Requests for a name are mapped to weighted buckets by the Memcached client library, and then fulfilled by the Memcached server. We use the default library, which permits rehashing of blocks in the event that a server becomes temporarily unavailable, but requires that the entire cache be flushed if bucket sizes or the number of nodes change. In our system, such cache flushes should not be common because we deploy clusters with a fixed number of nodes by design. Memcached has a built-in limit of 1 MB for each data block, and our client library uses this as the default maximum size for cached items; however, if the application specifies a larger limit, the client library transparently splits larger blocks in 1 MB chunks when storing, and reassembles them when retrieving them.

The disk cache service is implemented using MogileFS. MogileFS uses a MySQL database to store the locations where each file is stored. This single database quickly becomes a performance bottleneck when it is accessed for every cache read and write operation on the entire cluster. Our client library alleviates this problem by using Memcached to cache the file locations: as a result, most read operations do not involve querying the MySQL database.

For long term storage, we implement a simple service called Keep. Each warehouse instance with long term storage space runs a network server that accepts HTTP ``GET" and ``PUT" requests. The client library is responsible for replication, fault tolerance, and load balancing as described below.

A ``PUT" request is a signed request to copy data from cache to long term storage. The Keep server fetches the data itself from the local cache or from a remote cluster, according to the hints that come with the request. This approach is very efficient in the case where applications first store a lot of data in the disk cache, and later choose to keep some of that output in long term storage. This case has turned out to be so common that we have not yet implemented an operation that writes directly to long term storage.

When a "PUT" request results in a disk write, an accounting entry is also recorded, with the requestor's IP address and cryptographic signature. Currently, the server does not verify the signature because this has not been necessary in our environment, but it does demand that each request arrive in the form of a PGP signed message. It also records a timestamp and the full text of the request; and it requires that the message include a timestamp within 5 minutes of the server's system clock. We do not expect the overhead of checking signatures to cause an inordinate performance burden because of our large block size.

The client library uses MD5 checksums of data blocks to distribute them evenly among the available Keep servers. First, we specify that a given cluster has a fixed number of servers (there should be one on each cluster node). For a given block of data, we define eight preferred storage positions, derived from eight substrings of the block's 128-bit MD5 checksum: portions of the checksum are used to compute a list of eight different Keep servers, and these Keep servers are tried one by one until enough copies have been written.

When a block has been written to Keep, the client library notes which of the eight preferred Keep nodes were used, and encodes this information as a hexadecimal number, where the least significant bit corresponds to the first preferred storage position. The block location is given as the letter "K" (from "Keep") followed by the hexadecimal number, the symbol "@", and the name of the cluster. For example, if a block is stored on the templeton cluster using the second and third servers in the probe order, the list of locations is encoded as "K06@templeton". This string, along with the size of the block (an unusually short 3 byte block in this case) are appended to the MD5 checksum using the delimiter "+":

```
acbd18db4cc2f85cedef654fccc4a4d8+3+K06@templeton
```

This resulting block name, stored in a manifest, provides enough information for a Freegol or a batch job to

retrieve the block, regardless of which cluster it is stored on.

This notation provides an opportunity to support other storage systems – for example, “S” might designate blocks stored using Amazon's S3 service – and to cite multiple clusters and storage systems, when a block is stored in multiple locations.

A “GET” request is much like a request for a static HTML page: the MD5 hash provided in the client's request is the name of the disk file where the data is stored on the server. A Keep server may have up to four disks available for long term storage; in that case, it may have to perform four directory lookups in order to locate a file.

The storage controller maintains a database of manifest names and keys in a MySQL database. Any Freegol can connect to the storage controller's “warehoused” network server program and retrieve the key currently associated with a given name, or the entire list of names and keys. A Freegol can also submit a signed request to update the database by changing the key for a given name, or adding a new name. Currently, the “warehoused” program does not verify signatures of these requests because this has not been necessary in our environment, but it does demand that each update request arrive in the form of a PGP signed message, and that the Freegol correctly specify the key currently associated with the name.

The client library includes functions for reading and writing blocks to the cache and storage services. The library also provides convenient functions for constructing manifests while storing data, reading streams and individual files from an existing manifest, looking up manifest keys by name, and updating the name database. Command line tools are provided for submitting jobs, looking up details of current and completed jobs, reading and writing data sets, and copying data sets from one cluster to another. These programs also serve as examples of how to use the client library.

The warehouse client library makes it convenient for Freegols and batch jobs to read and write data on remote clusters as well as the default local cluster. We provide this flexibility – rather than requiring data to be explicitly copied from cluster to cluster using a separate mechanism – for several reasons. It allows applications to achieve various levels of data redundancy, either synchronously or asynchronously, according to their needs. It supports the convenience of running jobs on remote clusters without arranging for all of the required data to be copied ahead of time. Generally, it follows the trend of doing a reasonable thing by default while offering the flexibility to accommodate the diverse needs of a variety of applications.

3.4 Batch processing services

The “warehoused” program accepts signed job submissions from Freegols, and answers queries about previously submitted jobs. The “mapinit” program starts queued jobs when instances become available, using Slurm’s `salloc` command to reserve warehouse instances. At the start of each job, the “mrjobmanager” program first retrieves the specified version of the appropriate “mr-function” from a Subversion repository, then invokes it on one of the allocated warehouse instances. The mr-function examines the input object (normally a manifest) and instructs mrjobmanager to queue a number of job steps.

During the course of a job, mrjobmanager allocates individual job steps to warehouse instances, monitors their output and exit codes to detect failures, and re-queues them when they fail. Each job step is expected to store some output blocks and send the blocks’ names to mrjobmanager by printing them to its standard error file descriptor. When all job steps have completed, mrjobmanager reads these blocks and assembles them into a final output stream. This final output stream is expected to be a manifest, although this is not enforced. Finally, the database table is updated to reflect the output key (ie., a list of output blocks) and the time when the job finished.

While a job is running, mrjobmanager keeps the job table updated with the number of job steps in progress, finished, and remaining. These figures can be retrieved by any Freegol from the batch controller, and displayed to users as a progress indicator.

Order of execution and output assembly is controlled by job step numbers and level numbers. Step numbers begin at zero and are assigned sequentially by mrjobmanager; in the final stage of mrjobmanager the step numbers determine the order in which the job steps' output fragments are assembled into the final output stream. Level numbers can be used by mr-functions to control the order in which job steps are scheduled: a job step with level L will never begin until all job steps with level less than L have completed. Each job step can also be given a short input string; this can be used by the mr-function to keep track of which portion of the job each step is expected to compute, in case the job step number itself is not convenient for that purpose.

Typically, a mr-function completes step 0 by reading its input manifest and submitting one new job step for each stream in the manifest. Each of these job steps will read the input stream data, write output data in the form of a stream, store the stream description (one line of the manifest) as a short block in the cache, and report the hash of this short block to the job manager. When all job steps have finished, mrjobmanager looks up all of the individual job step hashes and assembles them, ordered by job step number, into one final output manifest. This final assembly step is inexpensive because it only involves lists of hashes and filenames; the job manager does not read or write any of the output data itself.

4 Applications and results

4.1 Freegols

We have implemented a few sample applications that demonstrate the warehouse client library and allow us to characterize performance.

The Genomator application is a storage/publication service. It currently allows users to browse and download images from a 300 gigabyte PMAGE data set [17]. In interactive mode, it converts images from TIFF to JPEG format and applies an ImageMagick "normalize" operation to increase contrast. This does not involve any batch processing, and the data set could have easily fit on a single disk; however, implementing the service as a Freegol gave us features like mirrored disks and scalability using existing hardware and staff resources.

The Regol application continuously re-schedules previously completed batch jobs when the cluster is idle, using the same inputs and parameters but substituting the current revision of the relevant mr-function. This helps us notice bugs as they are introduced into the source code repository. It is also a good source of information about performance characteristics of mr-functions, hardware configurations, and resource usage patterns. Regol is deployed on our "templeton" production cluster and is able to view and submit jobs on both clusters.

The Administrator web interface provides a generic job submission and monitoring interface. Users can select a mr-function and revision number, choose an input manifest from the list provided by the storage controller, and specify tunable parameters specific to the mr-function. Each of our clusters has its own Administrator web interface.

4.2 Mr-functions

The mr-functions we have implemented are generally concerned with problems in bioinformatics, specifically low cost DNA sequencing. Here we present some simpler applications that give a rudimentary illustration of the platform. Timing results from two of these can be found in Figure 5.

Figure 5: Timing figures from the templeton cluster show the effect of concurrency on per-process I/O speed. Mr-pivot-images shows diminishing returns as more instances are allocated: the data is read from the same set of disks regardless of allocation, and job speed is limited by the disk cache service's ability to handle many

concurrent readers. In contrast, mr-align-call's performance is nearly linear with larger node allocations.

```
\includegraphics[bb=110bp 0bp 841bp 595bp,clip,width=3.4in]
{4_data_personal_ward_Desktop_freelogy_svn_lyx_2008_usenix_figs_mr-pivot-images.eps}
\includegraphics[bb=110bp 0bp 841bp 595bp,clip,width=3.4in]
{5_data_personal_ward_Desktop_freelogy_svn_lyx_2008_usenix_figs_mr-aligncall.eps}
```

``mr-pivot-images'' reads a manifest with F images (one per frame position on a slide, typically about 2000) in each of C streams (one per imaging cycle, typically about 75) and outputs a manifest with C images in each of F streams. The structure of the input data is determined by the image acquisition process; the output structure is suitable for image alignment and analysis. This is a relatively inefficient operation because each of F job steps reads pieces from each of C streams. It performs 3.3 terabytes of I/O on our 300 gigabyte input data set. We will certainly want to optimize this if we intend to use it frequently; meanwhile it provides a convenient way to measure a cluster's performance under heavy I/O load.

``mr-aligncall'' reads each stack of images produced by ``mr-pivot-images'', analyzes and compares the images according to tunable parameters, and outputs short segments of DNA sequence as strings of A, C, G, and T characters.

``mr-zhash'' uncompresses its input (if compressed), computes hashes for individual files, and outputs text files similar to the output of the Linux ``md5sum'' command line tool. It is useful for determining whether two compressed data sets are equal when decompressed.

``mr-copy'' writes a copy of a data set read from a remote cluster (to make subsequent computation faster) or from the local cluster (to verify that all data is readable and passes checksum verification).

As the "mr-pivot-images" example suggests, we have found that the speed of an I/O-limited function is highly dependent on how closely the mr-function's operation corresponds to the way the data is arranged in the input manifest. Ideally, each step in a batch job reads all of its input data from a single stream, in the exact order in which it is processed. In the case of "mr-aligncall", this is easy to achieve because 2000 stacks of 75 2 MB images are processed by 2000 independent job steps. In contrast, for subsequent stages of our DNA sequencing applications, we spend much of our effort finding efficient ways to perform operations that are conceptually similar to "mr-pivot-images". In this sense, the manifest format is an expression of the performance characteristics of the storage service: if we design our workflows to cater to the structure of manifests, then we get the best performance from our system.

4.3 Storage tools

We use two command line tools to move data back and forth between the storage service (or cache) and a Freegol's local filesystem. "whput" copies a UNIX filesystem tree to the cache, stores the resulting manifest in the cache, and optionally attaches a name to the manifest via the storage controller. "whget" fetches a manifest from storage, downloads the blocks, computes the MD5 checksums of the individual files, and optionally writes the files to the local filesystem. These tools - and the warehouse client library in general - can be used from any host with access to TCP/IP ports on the warehouse instances, such as an administrator's workstation.

"whget.cgi" provides a web interface to the contents of a manifest. It allows users to view filenames and sizes, click individual files to download them, and download the manifest file itself. Along with Apache's mod_rewrite module, this makes it easy for a developer to selectively publish individual data sets.

4.4 Performance and reliability

Figure 6: "Regol" helps us notice problems by re-submitting selected jobs when the cluster is idle. This graph shows two different jobs being repeated. Solid shapes indicate successful jobs. Hollow shapes indicate failures.

Dashed lines indicate commits to the Subversion repository. If a string of failures begins at a vertical line and ends at another vertical line, it is most likely that the failures were caused by an errant commit.

```
\includegraphics[bb=40bp 0bp 761bp 429bp,width=3.5in]
{6_data_personal_ward_Desktop_freelogy_svn_lyx_2008_usenix_figs_final.eps}
```

In Figure 6 the repeatability of two jobs, ``mr-raw'' and ``mr-jp2'', is illustrated. These jobs are running simultaneously; they perform approximately one terabyte of input and output in each pair of runs. The ``jp2'' job uses the lossless JPEG2000 format to compress the image data and is CPU intensive. The ``raw'' job is primarily limited by I/O. By inspection it is clear that the cluster achieves more than 400 MB/s of I/O: 1 terabyte in total, divided by 2500 seconds for the slower job.

To further explore the aggregate I/O and computational capacity of both clusters, we ran a selection of ``mr-zhash'' cryptographic hash functions concurrently with the ``mr-pivot'' function described above. The input to mr-zhash is compressed data; in each job, the amount of data processed is over 100 times the amount read from cache. This mixture of computation-intensive and I/O-intensive work was repeated over a 16 hour period, using 42 instances on each cluster. Over the 16 hour period, ``mr-zhash'' processed 102 TB of uncompressed data (1.5 GB/s on 12 templeton instances, 380 MB/s on 12 uncle instances) while ``mr-pivot-images'' performed 74 TB of I/O (1 GB/s on 30 templeton instances, 290 MB/s on 30 uncle instances).

In the above test, we ran ``mr-zhash'' in sets of twelve concurrent jobs - for each of six hash functions, one job with the RAM cache enabled, and one without. The results are shown in Figure 7.

Figure 7: Mean time to read, uncompress, and run various checksum algorithms on 5.5 GB of compressed images, using an allocation of one warehouse instance per job on a busy cluster. Black bars represent jobs with the default client library configuration; hollow bars show the effect of disabling the RAM cache. Execution time is less predictable on uncle: some nodes are much slower than others, and the allocation of nodes to jobs is not

random, so we see artifacts in the first graph. Templeton's hardware is faster and more uniform; this is reflected in the second graph.

`\includegraphics[width=3in]{7_data_personal_ward_Desktop_freelogy_svn_lyx_2008_usenix_figs_uncle.eps}`

`\includegraphics[width=3in]{8_data_personal_ward_Desktop_freelogy_svn_lyx_2008_usenix_figs_temp.eps}`

Many of our design features are responses to lessons we have learned while using our two prototype clusters. For example, because we allocated space for cache on all of our disks before deploying the storage service, the storage service still shares disks with the cache. As a result, the storage service cannot serialize disk accesses. Our disk cache itself was deployed by adding a few disks at a time. This resulted in a poor distribution of data. The manifest used in the `mr-pivot-images` example has 50% of its blocks stored on only 22 disks. As we discussed above, performance suffers when a small number of disks are accessed by a larger number of concurrent processes. It is also noteworthy that even with four concurrent readers on one node, we currently achieve only 75 MB/s of I/O. We have verified with the UNIX `find` program that blocks read in 64 MiB chunks can be read from the file system at 60 MB/s. We have verified with the `iperf` program that our network can sustain 100 MB/s. Although we have only reached 75% of this limit, rather than the 90% we set out to achieve, we look forward to deploying a new cluster with a properly balanced disk cache and segregated disks for storage. Even if the storage service never surpasses 75 MB/s per node, we are confident that we can achieve 4–8 GB/s of aggregate I/O.

4.5 Utilization

Between November 2007 and March 2008 we completed 460 million seconds of computation (18% utilization) on templeton and 600 million seconds (24% utilization) on uncle. During this time we accumulated 30 TB of data in cache on templeton (consuming 60 TB, 98% of the disk space allocated to the cache service) and 10 TB of data on uncle (consuming 20 TB, 73% of the space allocated). The two clusters consist of new and old hardware costing \$150,000 in total. Annual costs include \$25,000 for floor space, power, cooling, and network service, and

about \$50,000 for staff costs.

If we had paid for this CPU time on a per-second basis at the rate charged by Amazon's EC2 for comparable instances – \$0.80 per hour for an extra large instance per node, \$0.20 per hour for two small instances for each of the 36 older uncle nodes – this would cost \$96,000 per year. Storing an average of 20 TB of data for the duration would cost an additional \$36,000 per year, at the Amazon S3 rate of \$0.15 per GB per month.

We can also consider the cost of the time spent copying data between S3 and EC2. Amazon does not specify the usable bandwidth between S3 and EC2, but if we assume that it is a very low 2 Gb/s, it costs \$41 to keep 47 EC2 nodes active while copying 1 TB of data from S3 to EC2. If our 25% utilization rate comes from working on 1 TB of data for one day every four days, the annual transfer cost is less than \$4000 per cluster. This cost is even lower if the available bandwidth is more than 2 Gb/s, which is likely. Therefore, in most cases we expect this transfer cost to be negligible compared to the cost of computation time and storage space.

At these rates, our two clusters will take three years to break even with an Amazon EC2 and S3 implementation. The discrepancy between these figures and those given in section 2.1 is a reflection of the lower number of CPU cores per node in our older hardware, as well as our low hosting costs. Even with this older hardware, a utilization level of 25% is enough to bring the break-even point down to two years.

5 Future Directions

For our projects – and we believe this is true for others too – it is difficult to budget for computation and storage needs. How much we want depends on how much it costs. A platform for universal personalized medicine should permit individuals to form small communities that suit their own needs, while retaining much of the economy of scale available to much larger communities. We believe that this can be achieved by building a highly decentralized global network of Free Factories that allocate underutilized resources through market

mechanisms.

Others have explored the possibility of capturing market signals from users and we believe this is an attractive way to allocate resources for our applications in the long term [4,10,26]. Since we have control of our architecture from the hardware up, we hope that implementation and experimentation with such mechanisms will provide an opportunity for fruitful future research.

Finally, in the spirit of free and open source software, we hope others will deploy Free Factories of their own for applications we have never imagined.

6 Acknowledgements

This work was supported by a Center for Excellence in Genome Sciences grant from the National Human Genome Research Institute. We would like to thank our anonymous reviewers for providing numerous insightful comments. The process of making revisions was greatly facilitated by our shepherd, John Wilkes. Art Mann at Silicon Mechanics helped us with our hardware related requests both large and small. Finally, we are in debt to many people who provided data and suggestions, and developed applications for Free Factories.

Bibliography

- 1 Amazon.com, Inc.
<http://aws.amazon.com/>.
- 2 Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler.
An analysis of latent sector errors in disk drives.
In SIGMETRICS '07, pages 289–300. ACM Press, 2007.

- 3 G. Bell, J. Gray, and A. Szalay.
Petascale computational systems.
IEEE Computer, 39(1):110–112, 2006.
- 4 John Brunelle, Peter Hurst, John Huth, Laura Kang, Chaki Ng, David C. Parkes, Margo Seltzer, Jim Shank, and Saul Youssef.
Egg: An extensible and economics-inspired open grid computing platform.
In GECON'06, Singapore, 2006. World Scientific Publishing.
- 5 Canonical Ltd.
<http://ubuntu.com>.
- 6 Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber.
Bigtable: a distributed storage system for structured data.
In OSDI'06, pages 205–218, Seattle, WA, 2006. USENIX Association.
- 7 G. M. Church.
The personal genome project.
Molecular Systems Biology, 1:2005.0030, 2005.
<http://personalgenomes.org>.
- 8 Coreboot.
<http://coreboot.org>.

- 9 Danga Interactive.
<http://danga.com>.
- 10 R.K. Dash, N.R. Jennings, and D.C. Parkes.
Computational-mechanism design: a call to arms.
Intelligent Systems, 18(6):40-47, 2003.
- 11 Jeffrey Dean and Sanjay Ghemawat.
MapReduce: Simplified data processing on large clusters.
In OSDI'04, pages 137-150, San Francisco, CA, 2004. USENIX Association.
- 12 Jon G. Elerath and Michael Pecht.
Enhanced reliability modeling of RAID storage systems.
In DSN'07, pages 175-184, 25-28 June 2007.
- 13 Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.
The Google file system.
In SOSP'03, pages 29-43, New York, NY, 2003. ACM Press.
- 14 GNU Project.
<http://gnu.org>.
- 15 Jim Gray, David T. Liu, Maria Nieto-Santisteban, Alex Szalay, David J. DeWitt, and Gerd Heber.
Scientific data management in the coming decade.

SIGMOD Record, 34:34-41, 2005.

- 16 Harvard University.
Free Factories source code, 2008.
<http://factories.freelogy.org>.
- 17 Jae Bum Kim, Gregory J Porreca, Lei Song, Steven C Greenway, Joshua M Gorham, George M Church, Christine E Seidman, and J. G. Seidman.
Polony multiplex analysis of gene expression (PMAGE) in mouse hypertrophic cardiomyopathy.
Science, 316(5830):1481-1484, Jun 2007.
- 18 Lawrence Livermore National Laboratory.
<https://computing.llnl.gov/linux/slurm/>.
- 19 Jeantine E Lunshof, Ruth Chadwick, Daniel B Vorhaus, and George M Church.
From genetic privacy to open consent.
Nature Reviews Genetics, 9:406-411, May 2008.
- 20 Nature Publishing Group.
Positively disruptive.
Nature Genetics, 40(2):119, Feb 2008.
- 21 Opendgear, Inc.
<http://opengear.com>.
- 22 OpenVPN, Inc.

<http://openvpn.net>.

- 23 Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz Andre Barroso.
Failure trends in a large disk drive population.
In FAST'07, San Jose, CA, 2007. USENIX Association.
- 24 Bianca Schroeder and Garth A. Gibson.
Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?
In FAST'07. USENIX Association, 2007.
- 25 Jay Shendure, Gregory J Porreca, Nikos B Reppas, Xiaoxia Lin, John P McCutcheon, Abraham M Rosenbaum, Michael D Wang, Kun Zhang, Robi D Mitra, and George M Church.
Accurate multiplex polony sequencing of an evolved bacterial genome.
Science, 309(5741):1728-1732, Sep 2005.
- 26 Jeffrey Shneidman, Chaki Ng, David C. Parkes, Alvin AuYoung, Alex C. Snoeren, Amin Vahdat, and Brent Chun.
Why markets could (but don't currently) solve resource allocation problems in systems.
In HotOS-X, Santa Fe, NM, 2005.
- 27 Software in the Public Interest.
<http://debian.org>.
- 28 Alexander Szalay and Jim Gray.
2020 computing: science in an exponential world.

Nature, 440(7083):413-414, Mar 2006.

29 X-Prize Foundation.

Archon x-prize for genomics.
<http://genomics.xprize.org/>.

30 Kun Zhang, Adam C Martiny, Nikos B Reppas, Kerrie W Barry, Joel Malek, Sallie W Chisholm, and George M Church.

Sequencing genomes from single cells by polymerase cloning.
Nature Biotechnology, 24(6):680-686, Jun 2006.

31 Kun Zhang, Jun Zhu, Jay Shendure, Gregory J Porreca, John D Aach, Robi D Mitra, and George M Church.

Long-range polony haplotyping of individual human chromosome molecules.
Nature Genetics, 38(3):382-387, Mar 2006.

About this document ...

Free Factories: Unified Infrastructure for Data Intensive Web Services

This document was generated using the LaTeX2HTML translator Version 2002-2-1 (1.71)

Copyright © 1993, 1994, 1995, 1996, Nikos Drakos, Computer Based Learning Unit, University of Leeds.

Copyright © 1997, 1998, 1999, Ross Moore, Mathematics Department, Macquarie University, Sydney.

The command line arguments were:

latex2html -split 0 -show_section_numbers -local_icons -no_navigation usenix.tex

The translation was initiated by Ward Vandewege on 2008-05-01
Ward Vandewege 2008-05-01

Augmenting RAID with an SSD for Energy Relief

Hyo J. Lee Computer Engineering Hongik University Seoul, Korea hjlee@mail.hongik.ac.kr

Kyu H. Lee Computer Science Purdue University West Lafayette, IN, 47907 kyuhlee@purdue.edu

Sam H. Noh School of Comp. & Info. Eng. Hongik University Seoul, Korea samhnoh@hongik.ac.kr

Abstract

In this paper, we describe a design of a novel architecture for RAID that uses an SSD as a large cache to conserve energy. This approach stems from the fact that short term footprints are small enough to be efficiently managed within an SSD. More specifically, in this study, we consider two simple approaches to reduce the energy consumed in RAID. First, when a read happens in RAID, a copy of the read request is copied to the SSD, so that future requests may be serviced by the SSD. Then, for writes, all writes are buffered in SSDs so that the

interval between requests may be increased reducing activities at RAID disks. We incorporate these approaches into a real implementation of a RAID 5 system that consists of four hard disks and an SSD in a Linux environment. Our preliminary results in actual performance measurements using the cello99 and SPC traces show that energy consumption is reduced by a maximum of 14%.

1 Introduction

The federal Environmental Protection Agency (EPA) reported electricity consumption of U.S. data centers in 2006 was about \$4.5 billion [2]. This is more than the electricity consumed by all color televisions in the U.S. These huge electricity bills are painful to the companies that run these data centers. For this reason and, more importantly, the environmental aspect, studies on energy conservation has become an active research issue. In this paper, we look into how energy can be conserved at the storage system component of a system, which is responsible for a large portion of energy consumption in today's data centers [17].

RAID is a popular form of storage system in data centers since they can provide high performance and fault-tolerance with relatively low cost. From the energy conservation perspective, however, it is more challenging due to the philosophy behind RAID. Performance of RAID comes from parallelism. RAID tries to even the load of every disk, and this forces every disk to be active even though the load may be light.

Recently, the Solid State Drive (SSD), a new type of disk device that makes use of Flash memory, is being spotlighted as a new storage medium due to its many attractive characteristics. It is fast, lightweight, durable, and noiseless, but most of all it is efficient in terms of energy. Despite these attractive characteristics, there are still a number of challenges that must be overcome for it to be adopted as a main storage component in data centers. Above all, the capacity of SSD is too small to construct data centers and the cost per capacity also can be too high for server systems.

In this paper, we describe the design of a novel architecture for RAID that uses an SSD as a large cache to conserve energy while meeting their performance goal. (Though any form of RAID is possible, we limit our study to RAID 5; hence hereafter, RAID will imply RAID 5.) If judiciously managed, this will allow the hard disks composing RAID to be inactive most of the time leading to energy savings. This approach stems from the fact that even though overall storage size is growing at a high rate, considerable space is being unused and even for used space, short term footprints are small enough to be efficiently managed within an SSD.

More specifically, in this study, we consider two simple approaches to reduce the energy consumed by RAID. For reads, when a read happens in RAID, a copy of the read request is copied to the SSD, so that future requests may be serviced by the SSD. Then, for writes, all writes are buffered in SSDs so that the interval between requests may be increased so that activities at RAID disks may be reduced.

We have implemented these approaches in a Linux environment with real hard disks and an SSD deployed. Our preliminary results in actual performance measurements using the cello99 and SPC traces show that energy consumption is reduced by a maximum of 14%, which is not significant, but encouraging. With many more optimizations possible, more convincing results are anticipated.

The rest of the paper is organized as follows. Section 2 discusses SSD basics and works related to this study. We then describe in more detail the approach we take in Section 3. Section 4 describes the implementation and the results. Finally, we summarize and give directions for future work in Section 5.

2 Background and Related Work

In this section, we first present an SSD overview and compare it with conventional disk drives. We then discuss several of the previously proposed energy management techniques for disk arrays.

1 SSD overview

Flash memory is widely used in many small devices and embedded systems because of its characteristics such as low energy consumption, fast data access, and light weight. Recently, many companies are producing and marketing Solid State Drives (SSDs) that are based on NAND Flash memory to adopt the many attractive Flash memory features into personal and server computers. NAND Flash memory-based SSDs are constructed from an array of Flash memory modules that are accessed in parallel. By employing parallelism and interleaving among the modules that comprise the SSD, performance of SSDs is considerably better than conventional hard disks for random reads, while sustaining comparable performance for other kinds of workloads. Performance of SSDs has been consistently increasing recently and is expected to continue increasing for some time in the future.

The greatest advantage of SSDs, though, is energy efficiency. Since SSDs do not have any mechanical moving parts energy consumption of SSDs is much smaller than disks. However at this time, SSDs have some limitations as server system storage. First, the capacity of SSD is not large enough for server computers. Currently, the 128GB SSD is the largest one available in the market, though surely in the close future, this is bound to increase. The more serious problem is that of cost per capacity; currently the cost per capacity of SSDs is no match to that of hard disks. Hence, though there have been approaches to construct RAID using only SSDs, this can be an attractive solution to the few tech savvy personal users but not for server systems in general.

2 Related works

There have been considerable work related to saving energy in the storage system. One approach that has been proposed is to use multispeed disks that have two or more rotational speed levels in active mode. Gurumurthi et al. [6] and Carrera et al. [5] propose a multi-speed disk that change their rotational speed dynamically, and they make use of this feature for energy consumption.

Zhu et al. [16] propose an energy management scheme that make use of two-speed disks for disk arrays. Unfortunately, to date, this type of disk is not yet readily available. Our approach is different from these

approaches in that we are making use of available technology, that is, SSDs, making our approach more practical.

Another common approach to save energy in conventional disk arrays is to keep disks in standby mode as long as possible. To achieve this goal, many different data management schemes have been proposed. Some previous studies have proposed migrating frequently accessed data to a particular disk(s) so that the other disks can remain in standby mode longer [15,16]. Some studies consider producing redundant data by making copies of the original data to some free disk area to prevent unnecessary spin up [8,11]. If the requested data is in a disk that is in standby mode and the replicated copy is in one of the active disks, the request may be serviced without spin up overhead.

Yet another common approach used for energy savings and improved performance is to make use of memory management algorithms for the cache on the controller. Here, redundant data is stored in memory located in the hardware disk array controller. Some studies have focused on prefetching schemes [3,13], while some studies have proposed caching schemes [4,9,17] for energy efficient disk arrays. However, because of the memory space limitation within the controller, the replacement policy becomes the critical issue for both caching and prefetching. Furthermore, it is very difficult, if not impossible, to expand memory in disk array controllers. Hence, use of this approach may be limited. In some sense, the approach taken in our study is similar to these approaches. First, we make use of redundant data that is kept in the SSD. Second, the SSD is used as a cache. Though conceptually the same, redundant data is kept in a much faster and energy efficient medium that does not require any spin up time. Furthermore, the SSD as a cache is much larger than could be imagined with the disk controller.

3 SSD cache in RAID

In this section we describe our idea in detail. First, we describe our observations that serve as our motivation. Then, we describe the design of our system.

1 Observation

Figure 1 shows two aspects of the HP cello99 trace, which is a popular I/O trace used in many I/O related studies. The dates of these traces are from November 29, Monday to December 4th, the Saturday of that week. The aspect that we need to note is the footprint of the trace relative to the whole data set size of the disk, denoted as the 'Footprint Ratio' in the figure. The numbers tell us that the actual footprint for each day is generally less than 0.2% of the whole data set. That is, given a day of activities, the range of blocks that are accessed is very small relative to the data that the disk possesses. The 'Footprint Size' on the right y-axis reveals the actual size of the footprint. It shows that over a day's activities, only 30GBs or less are being accessed. What is more, the dark column, which represents the newly inserted data each day in comparison to the data accessed the prior day is even smaller, always being well below 10GBs.

Figure 1: HP cello99 daily footprint ratio `\begin{figure} \centering \mbox{\subfigure[]{\epsfig{figure=footprint.eps, height=3in, angle=-90}} \quad } \end{figure}`

Our motivation for this work is based on these observations. That is, they tell us that if we could take this footprint, which is relatively quite small, into a more energy efficient medium and service requests from there, benefits may be possible. More specifically, if the footprint could be cached in a low energy medium allowing the high energy consuming RAID disks to reduce their activities, then energy conservation would be possible. Finding a cache of this size and satisfying the energy requirement is the question at hand.

There can be several candidates that may be able to satisfy this large cache requirement. The traditional hard disk is one such candidate. However, a hard disk is a high energy consumer, and if one disk is used to cache a whole day's activity, performance will obviously be degraded. Furthermore, the benefit of having a RAID, that is, high performance and reliability, simply disappears. Memory in the RAID controller could be another candidate.

This, however, is prohibitively expensive. Furthermore, RAM is volatile, hence the reliability of this method comes into question. The third candidate could be non-volatile RAM (NV-RAM), which resolves the volatility issue of RAM in RAID controllers. Since NV-RAM provides the memory interface, it would be easy to deploy them as a cache. However, NV-RAM is still very expensive and has limited capacity making it even more infeasible than RAM.

The medium that is energy efficient, non-volatile, large enough, and at the same time, able to provide high performance under reasonable cost today is the Solid State Drive (SSD). Due to these many merits, we employ the SSD as the large cache. Not only does SSDs consume low energy, while providing high performance, they are disks with the exact same disk interface as hard disks, hence incorporating them to RAID is simple. Only the controller needs to be aware of its existence.

2 System design

The system that we propose simply adds a single SSD to a normal RAID configuration. The key design issue is, then, how to cache the whole daily footprint so that all other disks may remain inactive. Occasionally, the RAID disks inevitably must awake. Ideally, this is when all dirty blocks residing in SSDs should be written and when all reads that are anticipated to happen in the future should be prefetched.

In the work that we present here, we are not able to provide all these features. Instead, we only consider a very simple approach. Specifically, when the very first read request of a block arrives at the controller, this read is satisfied at the RAID disk. The block, however, is copied to the SSD as well for possible later requests. For writes, all write requests are satisfied at the SSD to extend the RAID disk inactivity time. However, to prevent loss of data during SSD failure, we make a mirror copy in unused space of an active disk.

4 Evaluation

In this section, we discuss the evaluation of our system. We describe the implementation, the environment in which the evaluation was performed, and the results.

1 Evaluation environment

For our evaluation, we implement our system on the Linux software RAID environment. Our environment uses a SATA interface as we were not able to purchase SCSI SSDs. Though SATA based RAID would be slower than SCSI based RAID the idea presented here is equally applicable. As hard disks composing RAID, we use 500GB hard disks (ST3500320AS) produced by Seagate, and for the SSD cache, we use a 32GB SSD (MSD-SATA3035) produced by Mtron. According to their data sheets, this SSD consumes 0.5W when idle and 1.66–2.43W when reading/writing while the hddisk consumes 7.96–9.29W when idle and 11.16W when reading/writing [10,12]. All our implementations and evaluations are done on the Linux 2.6.21.7. The original RAID system consists of four hard disks and our proposed system adds one SSD to that original configuration.

The Linux software RAID is implemented in the md (multiple device) driver and controlled by the mdadm application. We modified the mdadm to consider the SSD cache. Furthermore, the following additions are made to the md device driver code.

We maintain a hash table to manage data in SSD. Along with the original disk number and sector number, we keep information such as it corresponding SSD sector number and its dirty/clean status in this hash table. When a read request arrives, the SSD is first checked through the hash table. If it is not found here, then the request is serviced via the RAID hard disk. While so doing, that sector is copied to the SSD and the hash table is updated. If the read request is found in the hash table, the request is serviced via SSD directly. If the request is a write, the request is written on the SSD and on a RAID hard disk that is awake, and again, the hash table is updated.

The performance measure of interest is energy and response time. To measure how much energy is consumed, we

take two approaches simultaneously. One is to implement a daemon that monitors the disk status. There are four statuses that our disks can be in, namely, sleep, standby, idle, and active. This daemon distinguishes between sleep, standby and idle/active (that is, it cannot distinguish active and idle states). At every 5 second intervals it checks to see what state the disk is in and records this into a separate disk aside from the RAID hard disks and the SSD. In the other approach, we make actual measurements of the energy being consumed using the HPM-300A, a power meter, which can measure power consumed by the entire system [1]. For the response time, we measure the time it takes for each request using the `gettimeofday` call before and after the requests are sent, driven by the workload.

For the workload we replay two traces. One is the cello99 traces that are widely used in I/O related research [7], and the other is a trace of the SPC Web search engine benchmark [14]. For the cello99 trace, three days worth of traces (dated Dec. 2 (Thursday), 3 (Friday), and 4 (Saturday)) are used. The former is write oriented, while the latter is a read oriented workload. For each of the workloads, we run the experiments at three different request rates representing the load of the system. To represent low, medium, and high loads, we generate 100, 200, and 500 requests per second, respectively. The requests are sent directly from the application to the md device.

2 Results

Figure 2: Energy consumption $\begin{matrix} \text{\scriptsize Cello99} \\ \text{\epsfig{figure=... ..} \\ \text{\epsfig{figure=energy_spc.eps, width=0.17\textwidth, angle=-90}} \end{matrix}$

Figure 3: Average response time $\begin{matrix} \text{\scriptsize Cello99} \\ \text{\epsfig{figure=... ..} \\ \text{\epsfig{figure=response_time_spc.eps, width=0.17\textwidth, angle=-90}} \end{matrix}$

The results of the experiments are presented in Figures 2 and 3. The former reports the energy consumed by the two different RAID systems. The x-axis reports the Watt-hours consumed during the run. The numbers reported here are the energy consumed by the disks. These were obtained by measuring the energy consumed by the system with the disks off, for the duration of the time that the trace would execute with the original RAID system, and then, subtracting this value from the actual energy measurements for executing the traces with the original RAID and the SSD attached systems. The disk off state was achieved by putting all the disks under the RAID configuration to the sleep state using the hdparm command. Note that more energy is being consumed for lower loads as it is taking longer. The energy savings at low loads is greater as well, saving approximately 14% for both the cello99 and SPC traces. For medium and high loads, the savings are minimal being less than 10%.

The savings reported here are only modest. However, they are encouraging in two aspects. First, through the disk monitoring daemon, we were able to observe that for most of the executions for both traces, the hard disks in the RAID system rarely sleep. This is because of the way the experiments are set up. We were not able to use the actual timing information available in the traces because our system is an actual implementation. Hence, we used the load approximations to inject requests to the system. This left no room for the disk to become idle, hence sleep time was reduced. We are currently investigating how to accurately model the request intervals of the traces so that a more realistic workload could be reproduced.

So one must wonder where all this savings is coming from. This is the second encouraging factor. In our current implementation, we are taking a simple and naive approach for making use of the SSD cache as described previously. Yet, the little activity that is being transferred to the SSD is resulting in around 10% savings in energy. Even though disks are not going to sleep, their activities are being reduced as some of these are being done by the SSD. This is resulting in the energy savings. With further optimizations such as intelligent prefetching schemes, further improvements should be possible.

Figure 3 shows the average response times observed by all the requests. For the SPC workload, which is largely composed of random reads, we see a roughly 17% improvement when the load is high and a small improvement for medium loads. For the low load, performance becomes worse by a small percent. For the cello99 trace, we see that response time increases by a maximum of 30% for the low load, though the other loads fair better. An interesting observation for this trace is that the response time decreases for higher loads. Though we are taking a closer look, we do not have a clear reason for this behavior though we can conjecture that the way writes are being serviced has something to do with it as the cello99 traces are write oriented and writes are being requested asynchronously.

5 Conclusion

In this paper, we proposed a design of a novel architecture for RAID that uses an SSD as a large cache to conserve energy while meeting their performance goal. This design was based on our observation that short term daily footprints are small enough and change slow enough to be efficiently managed within an SSD of today.

We incorporated our approaches into a real implementation of a RAID system using a Linux environment consisting of four hard disks and an SSD. Our preliminary results in actual performance measurements using the cello99 and SPC traces show that energy consumption is reduced by a maximum of 14%.

The performance numbers, though encouraging, were not satisfactory. We believe there is much room for improvement. In our current implementation, we took a simple and naive approach to making use of the SSD cache. In order to make better use of this cache, we need to incorporate a prefetching mechanism so that the RAID disks may remain idle for long durations. We also need to consider piggybacking reads and writes so that much of the disk activities may be done when and only when they are necessary. These, and more, are part of the research that are currently being conducted.

6 Acknowledgments

This work was supported by the Korea Science and Engineering Foundation(KOSEF) grant funded by the Korea government(MOST) (No. R0A-2007-000-20071-0)

Bibliography

- 1 HPM-300A.
<http://www.adpower21.com>.
- 2 Report to Congress on Server and Data Center Energy Efficiency Public Law 109-431.
<http://www.energystar.gov>, 2007.
- 3 S. H. Baek and K. H. Park.
Prefetching with Adaptive Cache Culling for Striped Disk Arrays.
In Proceedings of the Annual USENIX Technical Conference, 2008.
- 4 L. N. Bairavasundaram, M. Sivathanu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau.
X-RAY: A Non-Invasive Exclusive Caching Mechanism for RAIDs.
In Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA '04), 2004.
- 5 E. V. Carrera, E. Pinheiro, and R. Bianchini.
Conserving Disk Energy in Network Servers.
In Proceedings of the 17th International Conference on Supercomputing, 2003.
- 6 S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke.

DRPM: dynamic speed control for power management in server class disks.

In Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA '03), 2003.

7 HP Labs.

Tools and Traces.

<http://www.hpl.hp.com/research/ssp/software/>.

8 H. Huang, W. Hung, and K. G. Shin.

FS2: dynamic data replication in free disk space for improving disk performance and energy consumption.

In Proceedings of the Twentieth ACM Symposium on Operating Systems Principles (SOSP '05), 2005.

9 D. Li and J. Wang.

EERAID: energy efficient redundant and inexpensive disk array.

In Proceedings of the 11th ACM SIGOPS European Workshop (EW11), 2004.

10 Mtron.

MSD-SATA3035.

<http://mtron.net/>.

11 E. Pinheiro, R. Bianchini, and C. Dubnicki.

Exploiting redundancy to conserve energy in storage systems.

In Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '06), 2006.

12 Seagate.

ST3500320AS.

<http://www.seagate.com/>.

- 13 S. W. Son and M. Kandemir.
Energy-aware data prefetching for multi-speed disks.
In Proceedings of the 3rd ACM Conference on Computing Frontiers (CF '06), 2006.
- 14 Storage Performance Council.
SPC-1 Specification.
<http://www.storageperformance.org/specs>.
- 15 C. Weddle, M. Oldham, J. Qian, A.-I. A. Wang, P. Reiher, and G. Kuenning.
PARAID: a gear-shifting power-aware RAID.
In Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST '07), 2007.
- 16 Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes.
Hibernate: helping disk arrays sleep through the winter.
In Proceedings of the Twentieth ACM Symposium on Operating Systems Principles (SOSP '05), 2005.
- 17 Q. Zhu, F. M. David, C. F. Devaraj, Z. Li, Y. Zhou, and P. Cao.
Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management.
In Proceedings of the 10th International Symposium on High Performance Computer Architecture (HPCA '04), 2004.

About this document ...

Augmenting RAID with an SSD for Energy Relief

This document was generated using the LaTeX2HTML translator Version 2008 (1.71)

Copyright © 1993, 1994, 1995, 1996, Nikos Drakos, Computer Based Learning Unit, University of Leeds.

Copyright © 1997, 1998, 1999, Ross Moore, Mathematics Department, Macquarie University, Sydney.

The command line arguments were:

```
latex2html -split 0 -show_section_numbers -local_icons -no_navigation HotPower08.tex
```

The translation was initiated by on 2008-11-12

2008-11-12

Energy Aware Consolidation for Cloud Computing

Shekhar Srikantaiah

Copyright © 2004 ~ 2009 Hometown Innovation Automation Inc.

Web Site: <http://www.hwswworld.com> Tel: 86-21-51978298 Fax: 86-21-51978299

5th Fl., Jingang Rd., Shanghai 2001206, China

Pennsylvania State University

Aman Kansal

Microsoft Research

Feng Zhao

Microsoft Research

Abstract

Consolidation of applications in cloud computing environments presents a significant opportunity for energy optimization. As a first step toward enabling energy efficient consolidation, we study the inter-relationships between energy consumption, resource utilization, and performance of consolidated workloads. The study reveals the energy performance trade-offs for consolidation and shows that optimal operating points exist. We model the consolidation problem as a modified bin packing problem and illustrate it with an example. Finally, we outline the challenges in finding effective solutions to the consolidation problem.

1 Introduction

One of the major causes of energy inefficiency in data centers is the idle power wasted when servers run at low utilization. Even at a very low load, such as 10% CPU utilization, the power consumed is over 50% of the peak

power [1]. Similarly, if the disk, network, or any such resource is the performance bottleneck, the idle power wastage in other resources goes up. In the cloud computing approach multiple data center applications are hosted on a common set of servers. This allows for consolidation of application workloads on a smaller number of servers that may be kept better utilized, as different workloads may have different resource utilization footprints and may further differ in their temporal variations. Consolidation thus allows amortizing the idle power costs more efficiently.

However, effective consolidation is not as trivial as packing the maximum workload in the smallest number of servers, keeping each resource (CPU, disk, network, etc) on every server at 100% utilization. In fact such an approach may increase the energy used per unit service provided, as we show later.

Performing consolidation to optimize energy usage while providing required performance raises several concerns. Firstly, consolidation methods must carefully decide which workloads should be combined on a common physical server. Workload resource usage, performance, and energy usages are not additive. Understanding the nature of their composition is thus critical to decide which workloads can be packed together. Secondly, there exists an optimal performance and energy point. This happens because consolidation leads to performance degradation that causes the execution time to increase, eating into the energy savings from reduced idle energy. Further, the optimal point changes with acceptable degradation in performance and application mix. Determining the optimal point and tracking it as workloads change, thus becomes important for energy efficient consolidation.

This paper exposes some of the complexities in performing consolidation for power optimization, and proposes viable research directions to address the challenges involved. We experimentally study how performance, energy usage, and resource utilization changes as multiple workloads with varying resource usages are combined on common servers. We use these experimental insights to infer optimal operating points that minimize energy usage with and without performance constraints. We then discuss consolidation as a modified multi-dimensional bin-

packing problem of allocating and migrating workloads to achieve energy optimal operation. To concretely illustrate this research direction, we present a computationally efficient heuristic to perform consolidation in a simplified scenario. There are many issues that affect consolidation, including server and workload behavior, security restrictions requiring co-location of certain application components, and power line redundancy restrictions. The paper focuses only on a manageable but important subspace spanned by CPU and disk resource combinations. The many issues that need to be addressed to achieve a real-world implementation of such consolidation methods are also discussed to help point out fruitful research directions.

2 Understanding Consolidation

Understanding the impact of consolidating applications on the key observable characteristics of execution, including resource utilization, performance, and energy consumption, is important to design an effective consolidation strategy.

To explore this impact, we used the experimental setup shown in Figure 1. A cloud consisting of $m = 4$ physical servers hosting k controlled applications services requests from controlled clients. Each client generates requests at a desired rate. Each request consists of jobs with specified levels of resource usage on each server resource (here, each server is considered as comprising of two resources: processor and disk). Each server is connected to a power meter (WattsUp Pro ES) to track power consumption. The Wattsup meter samples power at a maximum rate of 1 Hz, and hence any desired utilization state must be sustained for more than 1 second. We designed the request processing to maintain a uniform utilization state for 60 seconds and averaged the readings over this period to get accurate energy data. The resource utilization is tracked using the operating system's built in instrumentation accessed through the Xperf utility to track processor and disk usage.

Figure 1: Experimental setup. `\begin{figure} \centering \epsfig{file=expsetup.eps,angle=-90,width=0.45\textwidth} \end{figure}`
`...degraphics[width=0.45\textwidth][expSetup.pdf] \vspace{-0.15in} \end{figure}`

Consolidation influences utilization of resources in a non-trivial manner. Clearly, energy usage does not linearly add when workloads are combined, due to a significant percentage of idle energy. But utilization and performance also change in a non-trivial manner. Performance degradation occurs with consolidation because of internal conflicts among consolidated applications, such as cache contentions, conflicts at functional units of the CPU, disk scheduling conflicts, and disk write buffer conflicts.

Figure 2: Performance degradation and energy consumption with varying combined CPU and disk utilizations.
`\begin{figure} \begin{center} \subfigure[Performance degradation]{ %\inclu... ...space{-0.2 in} \end{center} \vspace{-0.2 in} \vspace{-0.2 in} \end{figure}`

To study the impact of consolidation with multiple resources, we measured performance and energy while varying both CPU and disk utilizations. First, an application with 10% CPU utilization and 10% disk utilization is started. Then, it is combined with workloads of varying CPU and disk utilizations, ranging from 10% to 90% in each resource. The results are plotted in Figure 2(a). The performance degradation observed along the CPU utilization axis is not as significant as that observed along the disk utilization axis implying that increasing disk utilization is a limiting factor for consolidated performance on this server.

Energy consumption per transaction of a consolidated workload is influenced by both resource utilization and performance. Typical variation of energy per transaction with utilization (of a single resource) can be expected to result in a "U"-shaped curve. When the resource utilization is low, idle power is not amortized effectively and hence the energy per transaction is high. At high resource utilization on the other hand, energy consumption is high due to performance degradation and longer execution time.

Figure 2(b) plots the energy consumption per transaction of the consolidated workload, for the same scenario. We can make a few important observations from this result. Firstly, energy consumption per transaction is more

sensitive to variations in CPU utilization (as seen by the deeper "U" along the CPU axis) than variations in disk utilization as seen by the relatively flat shape of the curve along the axis of disk utilization. Secondly, there exists an optimal combination of CPU and disk utilizations where the energy per transaction is minimum. This occurs at 70% CPU utilization and 50% disk utilization for this setup, though the numbers could depend on the specific machines and workloads used. This energy optimal combination may vary if we further impose bounds on the performance degradation tolerable to us, but we can always find an optimal combination of resource utilizations that minimizes energy per transaction taking into consideration both the resources. Note also that this optimal point may be significantly different than that determined by considering each resource in isolation.

3 Consolidation Problem

The goal of energy aware consolidation is to keep servers well utilized such that the idle power costs are efficiently amortized but without taking an energy penalty due to internal contentions.

The problem of loading servers to a desired utilization level for each resource may be naïvely modeled as a multi-dimensional bin packing problem where servers are bins with each resource (CPU, disk, network, etc) being one dimension of the bin. The bin size along each dimension is given by the energy optimal utilization level. Each hosted application with known resource utilizations can be treated as an object with given size in each dimension. Minimizing the number of bins should minimize the idle power wastage. However, that is not true in general, causing the energy aware consolidation problem to differ from traditional vector bin packing:

Performance degradation: Unlike objects packed in bins, the objects here have a characteristic not modeled along the bin dimensions: performance. When two objects are packed together, their performance (such as throughput) degrades. Since performance degradation increases energy per unit work, minimizing the number of bins may not necessarily minimize energy. This aspect must be accounted for in the solution.

Power variation: In a traditional bin packing problem, if the minimum number of bins is n , then filling up $n-1$ bins completely and placing a small object in the n -th bin may be an optimal solution. Here, even given

the optimal number of bins, the actual allocation of the objects may affect the power consumed.

Designing an effective consolidation problem thus requires a new solution. The actual implementation must also address several other challenges discussed in section 4. Before discussing those issues, we consider an example heuristic for consolidation, to help illustrate the problem concretely (though not necessarily providing a final or optimal solution).

3.1 Consolidation Algorithm

This algorithm aims to find a minimal energy allocation of workloads to servers. Minimum energy consumption occurs at a certain utilization and performance. However, that performance may not be within the desired performance constraints and hence we design the algorithm to minimize energy subject to a performance constraint. This constraint also helps account for the first difference mentioned above from traditional bin-packing: if we set the performance constraint to that needed for optimal energy consumption, the algorithm will not attempt to minimize the number of bins beyond the number that minimizes power. If the performance constraint is tighter than that required for optimal energy, the number of bins is automatically never reduced below the optimal.

Also, while energy and performance change in a non-trivial manner when workloads are combined, it is worth noting that the resource utilizations themselves are approximately additive, especially at utilizations that are lower than the optimal levels for each resource. Using utilizations as the bin dimensions thus allows our algorithm to operate with a closed form calculation of bin occupancy when combining objects.

The algorithm proceeds as follows: The first step determines the optimal point, from profiling data such as shown in Figure 2(b). Next, as each request arrives, it is allocated to a server, resulting in the desired workload distribution across servers. The crux lies in using a simple, efficient and scalable heuristic for bin packing. The

heuristic used here maximizes the sum of the Euclidean distances of the current allocations to the optimal point at each server. This heuristic is based on the intuition that we can use both dimensions of a bin to the fullest (where "full" is defined as the optimal utilization point) after the current allocation is done, if we are left with maximum empty space in each dimension after the allocation. The n -dimensional Euclidean distance (Δ_e) provides a scalar metric that depends on the distances to the optimal point along each dimension. If the request cannot be allocated, a new server is turned on and all requests are re-allocated using the same heuristic, in an arbitrary order. This approach is adaptive to changing workloads (temporal variations in requests) as we allocate newer requests to appropriate servers. It also works seamlessly in presence of heterogeneity: in that case the optimal combinations would be different for each type of server.

To illustrate the heuristic, consider the simple example shown in Table 1. The current status of two servers is shown: server A is currently executing at [30, 30], which represents 30% CPU utilization and 30% disk utilization, and Server B is at [40, 10]. A new request with workload requirement [10,10] is to be allocated. Suppose the optimal point for both A and B is [80, 50]. If the application is allocated to A, the sum of the Euclidean distances to the respective optimal points, $\Delta_e^A([40,40] - [80,50]) + \Delta_e^B([40,10] - [80,50]) = 97.8$, is greater than that achieved by allocating the request to B. Thus the request is allocated to A, leaving A running at [40, 40] and B at [40, 10].

Table 1: Scenario to illustrate proposed heuristic.

	CPU	Disk	Opt_CPU	Opt_Disk	Δ_e	$\sum\{\Delta_e\}$
A_orig	30	30	80	50	53.8	97.8
A_after	40	40	80	50	41.2	
B_orig	40	10	80	50	56.6	96.2
B_after	50	20	80	50	42.4	

Note that an optimal algorithm may find better allocations. In the example above, as a result of the allocation chosen, it is feasible to allocate subsequent requests such as [10, 40], [20, 40], [30, 40], [40, 30], and [40, 40], but not requests such as [50, 10], [50, 20], which would have been feasible if the current request of [10,10] was allocated to B. At each decision point the heuristic chooses an allocation that would leave the system capable of servicing the widest range of future requests. An optimal scheme however, would choose the allocations based on all requests simultaneously rather than allocating the current request without changing existing allocations. In general, while it is possible to develop better heuristic algorithms, the algorithm presented here serves as an illustration to show that energy savings are possible over the base case. We reserve a more detailed study of the pathological cases for this algorithm (like extremely random disk accesses) for future work.

The optimal scheme may have a very high computational overhead, making it infeasible to be operated at the request arrival rate. However, for the purpose of comparison, we implemented the optimal scheme based on an exhaustive search of all possible allocations for a very small cloud consisting of four servers.

Table 2: Various mixes of applications considered for our multiprogrammed workload.

	# Apps	Total CPU utilization	Total disk utilization
Mix1	6	84.87	85.86
Mix2	6	93.72	53.87
Mix3	6	78.79	150.58
Mix4	6	91.37	108.92

3.2 Experimental Evaluations

Table 2 shows the characteristics of various mixes of applications used in the comparison of the example heuristic with the optimal. The application mixes represent a range of consolidated workloads. Mix1 and Mix4 have equal

CPU and disk utilizations but vary in the absolute total resource used. Mix2 and Mix3 are skewed toward high CPU utilization and high disk utilization respectively. The energy consumption per transaction of consolidated workloads is experimentally measured using the setup of Figure 1.

Figure 3: Comparison of our heuristic scheme with the optimal scheme. $\begin{figure} \vspace{-0.3in} \begin{center} \subfigure[{\$tolerance = 20\%... ..perInfty.pdf}\vspace{-0.3in}] \end{center} \vspace{-0.3in} \end{figure}$

Energy per transaction with a maximum performance degradation tolerance of 20% and infinite tolerance are plotted in Figures 3(a) and 3(b) respectively. Clearly, greater energy savings are achieved by both schemes at higher tolerance in performance degradation. The energy used by the proposed heuristic is about 5.4% more than optimal (not considering the energy spent on computing the optimal) on an average at 20% tolerance. It approaches the optimal at infinite tolerance in performance degradation for the test scenario.

4 Discussion

The consolidation problem discussed above helps appreciate some of the key aspects of the performance- utilization- energy relationship. Our ongoing effort in designing a solution has helped uncover several complications, not addressed in the illustrative algorithm above, that lead to a rich set of research issues.

Multi-tiered Applications: Unlike the application described in the previous section, a realistic application may use multiple servers such as a front-end request classification server, a back-end database server to access requested data, an authentication server to verify relevant credentials and so on. Such an application is commonly referred to as a multi-tiered application. Each tier has a different resource footprint and the footprints change in a correlated manner as the workload changes. The consolidation problem involves allocating the multiple tiers of all applications across a subset of available physical servers such that each application may operate at its desired performance level with minimum total energy usage. The allocation may have to be adapted dynamically as

workloads evolve over time, either via intelligent allocation of new requests, or via explicit migration of existing state from one physical server to another.

Composability Profiles: We showed how the performance, resource utilization, and energy vary as multiple applications are consolidated on a common server with varying workload serviced. The utilization and performance was tracked using Xperf and power using an external power meter. In a real cloud computing scenario, the overhead of running performance tracking and deploying external power meters may be unacceptable and better alternatives may be needed [4]. Also, measuring the energy-performance relationship itself is non-trivial as the number of applications and the number of possible combinations of their multiple tiers, across multiple server types, may be very large. The use of extra running time for profiling of an application's performance and energy behavior with varying mixes may not be feasible due to cost reasons. Run time variations may even render profiles less relevant. Thus, consolidation research also involves developing efficient methods to understand the composability behavior of applications at run time.

Migration and Resume Costs: While the example in the previous section assumed an application serving small stateless requests, that is not the only type of applications. Other applications may maintain a significant persistent state, such as a long lived chat session [1] or a video streaming session. Such applications involve a non-trivial overhead in migration from one machine to another. Migration may involve initiating the required configuration and virtual machine needed for the migrating application, incurring additional costs. Also, the dynamic nature of allocation implies that the sleep and wake-up costs of the servers need to be considered. When multiple servers are running at low utilization due to workload dwindling off, their applications may be consolidated to fewer servers allowing some physical servers to be set to sleep mode. However, the migration costs and sleep/wake-up costs may sometimes overshoot the benefit from shutdown of servers. Efficient methods to dynamically adapt to the workload variations with realistic migration and server resume costs for different categories of applications is an interesting problem.

Server Heterogeneity and Application Affinities: We initially assumed that any application may be hosted on any machine. However, in practice, certain applications may have affinities for certain physical resources. For instance, it may be more efficient to run a database server directly on the physical server that hosts the disk drives, since the database software may be optimized for the special read-write characteristics of disks and may directly access the raw storage sectors on the disks. Running the database over network connected storage with additional intermittent layers of storage drivers and network file system abstractions may make it inefficient. Aside from configuration, servers in a cloud facility may vary in energy-performance characteristics due to age of hardware. These variations are crucial to be exploited in consolidation as the energy usage of an application would depend on which server is used and in what application mix. The consolidator may not always run an application on the server most efficient for it since a global optimization is performed. Migration and resume costs may restrict the application from migrating to a more efficient server. Ensuring globally optimal operation in a dynamic and heterogeneous setting is an important problem.

Application Feedback: Our performance-energy study controlled the workload to measure energy and performance, assuming constant quality of service is provided at each performance level. However, certain applications may change their behavior in response to resource availability. For instance, a streaming application may be designed to lower the video streaming rate based on system load, the network stack may have built in congestion control, and so on. Consolidation may reduce resource availability by increasing utilization, incurring an acceptable level of performance degradation to save energy. However, if the application changes its behavior in response to resource availability, such feedback may result in complex behavior that makes it hard to guarantee the performance constraints. Designing consolidation methods for such applications or designing applications to be consolidation-friendly is an interesting research challenge.

Additionally, the example methods discussed here only considered the CPU and disk resources. Other resources

such as network and memory should also be considered, as these may be the bottleneck resources for certain applications. Operational constraints such as co-location of certain data and components for security, power line redundancies, and cooling load distribution may affect the design as well. Consolidation methods may need to account for several variations such as the multiple types of requests served by an applications – their expected processing times and the exact set of tiers used to service them. Some of the applications may use external services, not hosted by the cloud operator, further making the relationship more complex.

5 Related work

Consolidation of resources and controlling resource utilizations has been researched in recent times. A framework for adaptive control of virtualized resources in utility computing environments is presented in [5]. In the past, cluster resource management has posed similar problems to that of consolidation of virtualized resources. For instance, [7] proposes a two level resource distribution and scheduling scheme, and [8] proposes a cluster level feedback control scheme for performance optimization. Power aware load balancing and migration of activity within a processor, among multicore processors and servers has also been studied in [3,2,6]. However, there has been little work on joint power and performance aware schemes for multi-dimensional resource allocation. Specifically, characterizing power-performance characteristics of consolidated workloads with respect to multiple resources has not been adequately addressed.

Bibliography

- 1 CHEN, G., et al.
Energy-aware server provisioning and load dispatching for connection-intensive internet services.
In NSDI (2008).
- 2 GOMAA, M., et al.

Heat-and-run: leveraging smt and cmp to manage power density through the operating system.
SIGOPS Oper. Syst. Rev. 38, 5 (2004).

3 HEO, S., BARR, K., AND ASANOVIĆ, K.

Reducing power density through activity migration.
In ISLPED (2003).

4 KANSAL, A., AND ZHAO, F.

Fine-grained energy profiling for power-aware application design.
In Workshop on Hot Topics in Measurement and Modeling of Computer Systems (2008).

5 PADALA, P., et al.

Adaptive control of virtualized resources in utility computing environments.
In European Conference on Computer Systems (2007).

6 RANGANATHAN, P., et al.

Ensemble-level power management for dense blade servers.
In ISCA (2006).

7 SHEN, K., et al.

Integrated resource management for cluster-based internet services.
SIGOPS Oper. Syst. Rev. 36, SI (2002), 225-238.

8 WANG, X., AND CHEN, M.

Cluster-level feedback power control for performance optimization.
In HPCA (2008).

About this document ...

Energy Aware Consolidation for Cloud Computing

This document was generated using the LaTeX2HTML translator Version 2002-2-1 (1.71)

Copyright © 1993, 1994, 1995, 1996, Nikos Drakos, Computer Based Learning Unit, University of Leeds.

Copyright © 1997, 1998, 1999, Ross Moore, Mathematics Department, Macquarie University, Sydney.

The command line arguments were:

```
latex2html -split 0 -show_section_numbers -local_icons -no_navigation main.tex
```

The translation was initiated by Shekhar Srikantaiah on 2008-11-14

Shekhar Srikantaiah 2008-11-14

Sonic Nirvana: Using MEMS Accelerometers as Acoustic Pickups in Musical Instruments

By Rob O' Reilly, Alex Khenkin, and Kieran Harney

Introduction

MEMS¹ (microelectromechanical systems) technology builds on the core fabrication infrastructure developed for silicon integrated circuits. Micromechanical structures are created by etching defined patterns on a silicon substrate to form sensor elements or mechanical actuators that can move fractions of a micron. Pressure sensors, one of the first high volume MEMS applications, now monitor pressure in hundreds of millions of engine manifolds and tires; and MEMS accelerometers have been used for over 15 years for airbag deployment, rollover detection, and automotive alarm systems.

MEMS accelerometers² are also used for motion sensing in consumer applications, such as video games and cell phones. MEMS micromirror optical actuators are used in overhead projectors, HDTVs, and digital theater presentations. In recent years, MEMS microphones³ have begun to proliferate the broad consumer market, including cell phones, Bluetooth headsets, personal computers, and digital cameras.

This article describes some of the key technologies deployed in MEMS accelerometer products and discusses how this technology can bring a new dimension to acoustic transducers.

MEMS Accelerometer Technology

The core element of a typical MEMS accelerometer is a moving beam structure composed of two sets of fingers: one set is fixed to a solid ground plane on a substrate; the other set is attached to a known mass mounted on springs that can move in response to an applied acceleration. This applied acceleration (Figure 1) changes the capacitance between the fixed and moving beam fingers.⁴

Figure 1. MEMS accelerometer structure.

Figure 2. ADXL50 MEMS accelerometer structure.

The dimensions of these MEMS structures, on the order of microns (Figure 2), require very high precision silicon photolithography and etching process technologies. MEMS structures are typically formed from single-crystal silicon, or from polysilicon that is deposited at very high temperatures on the surface of a single-crystal silicon wafer. Structures with very different mechanical characteristics can be created using this flexible technology. One mechanical parameter that can be controlled and varied is spring stiffness. The mass of the sense element and the damping of the structure can also be modified by design. Sensors can be produced to measure fractions of one g or hundreds of g's with bandwidths as high as 20 kHz.

Figure 3. ADXL202 ± 2 g accelerometer.

The MEMS sensing element can be connected to the conditioning electronics on the same chip (Figure 3) or on a separate chip (Figure 4). For a single-chip solution, the capacitance of the sense element can be as low as 1 to 2 femtofarads per g, which equates to measurement resolution in the attofarad range! In a two-chip structure, the capacitance of the MEMS element must be high enough to overcome the parasitic capacitance effects of the bond wires between the MEMS and the conditioning ASIC (application specific integrated circuit).⁵

Figure 4. Cross-section of a typical two-chip accelerometer.

Accelerometers as Vibration Measurement Sensors

The concept of using vibration sensing transducers as acoustic pickups in musical instruments is not new.⁶ Piezo- and electromagnetic transducers are the basis for many of today's acoustic pickup applications. Tiny MEMS accelerometers are so small and low in mass that they have no mechanical or mass loading effect on the instrument, making them attractive for these applications; but to date their use has been limited due to the narrow bandwidth of commercially available acceleration sensors.

Some recent breakthroughs in accelerometer technology have enabled the production of very small accelerometers with very wide bandwidth. The ADXL0017 (Figure 5) high-g (± 70 g to ± 500 g), single-axis accelerometer has 22-kHz bandwidth and comes in a 5-mm \times 5-mm \times 2-mm package. This is ideal for monitoring vibration to determine the state-of-health of motors and other industrial equipment by detecting changes in their acoustic characteristics. In the early stages of bearing wear, a clear vibration signature that develops in the audio band can be detected with a high-g vibration sensor attached to the system housing. This particular sensor, which measures acceleration on the order of 10s of g's, is not sensitive enough for use as an acoustical vibration sensor for musical instruments. Also, it only senses along one axis of motion, while an ideal acoustic sensor will measure the response along all three axes. It does demonstrate, however, that full audio bandwidth acceleration transducers can be produced using MEMS technology.

Figure 5. ADXL001 Frequency response curve.

Low-g accelerometers can measure acceleration down to milli g's, but are typically bandwidth-limited around 5 kHz. This limitation may be associated with the fact that few commercial applications require significant

bandwidth (the primary applications involve the detection of human motion or gravity-driven acceleration), so there has been little motivation to develop sensors suited specifically for audio band measurement.

A 3-axis accelerometer has three separate outputs that measure acceleration along the Cartesian X, Y, and Z axes.

The ADXL3308 3-axis, low-g accelerometer has wider effective bandwidth than other traditional low-g accelerometers. Its bandwidth is up to 6 kHz on X- and Y axes, and around 1 kHz on the Z axis. While not ideal, this expanded bandwidth allows the part to gather useful information in the audio band. The output is analog, so it can be easily instrumented and used with standard audio recording equipment. Housed in a standard surface-mount package, it takes advantage of the mature semiconductor manufacturing infrastructure. Measuring less than $4\text{ mm} \times 4\text{ mm} \times 1.45\text{ mm}$ (Figure 6), the product can fit into places unimaginable with traditional accelerometer technology. Its very small size does not cause mass loading or other changes in the response of the system being measured. Later we will explore how this low-g accelerometer can be applied as an acoustic pickup for a guitar.

Figure 6. MEMS accelerometer, $4\text{ mm} \times 4\text{ mm} \times 1.45\text{ mm}$.

Acoustic Feedback

Beginning with the introduction of omnidirectional condenser and dynamic microphones in the mid 1920s by Søren Larsen, the Danish scientist who first discovered the principles of audio feedback (known as the Larsen effect), acoustic feedback has been a demon few audio engineers are able to totally control, making it unavoidable in live sound. The Beatles experimented with this audio artifact, then decided to add it to their memorable introduction to “I Feel Fine” in 1964.10 Rock ’ n’ Roll then set out to tame the beast by embracing it, making acoustic feedback a striking characteristic of rock music. Electric guitar players such as Pete Townshend and Jimi Hendrix deliberately induced feedback by holding their guitars close to the amplifier. As the fad waned, audio engineers continued their struggle with acoustic feedback’s undesirable ear-shattering effects, particularly in

live sound applications. In the perfect world of a well-appointed and acoustically treated recording studio, a high-end omnidirectional microphone will record instruments with an astonishing degree of realism and fidelity. Artists who know and cherish this sound have long sought the ability to reproduce it on stage. Although recording a live show with studio sound quality is every musician's dream, it has been virtually impossible. Even if sound reinforcement rigs sounded good, arenas had excellent acoustics, and sound engineers knew everything there was to know about mixing sound and had the best gear available, there would still remain one obstacle on the road to sonic nirvana: feedback.

Acoustic Pickups

Acoustic feedback is typically minimized by using directional microphones. This works to a certain extent, but requires constant management by sound engineers to adapt to the changing characteristics of a stage venue.

Musical instruments can be amplified using pickups. The technologies vary, but the basic idea is to sense the vibrations of the instrument's body directly, rather than the sound it produces in the air. The advantage is obvious: these pickups generate almost no acoustic feedback as they are not sensitive to airborne sound. The shortcomings are many: finding a good-sounding location on an instrument body is notoriously difficult, the sonic characteristics of piezo pickups are far from perfect, and their high output impedance requires special instrument inputs or direct boxes. In addition, they can be large and can interfere with the natural acoustic behavior of the instrument.

This leads to the idea of a low-mass contact microphone. Suppose that we used a surface transducer that measured the acceleration of the instrument's body, preferably on more than one axis.¹¹ This transducer would have good linearity and be so lightweight that it would not acoustically affect the instrument being measured.

Suppose further that the transducer had similar output level, output impedance, and power requirements as a traditional microphone. In short, suppose that a musician could just plug this transducer into a microphone preamp

or mixer input, just like any other microphone.

Contact Microphones

An attentive reader will notice the mention of acceleration in the preceding paragraph. Our ears respond to sound pressure, so microphones are designed to sense sound pressure. To simplify matters greatly, the sound pressure in the immediate vicinity of a vibrating body is proportional to acceleration.¹² What if an accelerometer had enough bandwidth to be used as a contact microphone?

To explore this concept, a 3-axis accelerometer was mounted on an acoustic guitar to act as a pickup. The vibration of the instrument was measured and compared to the built-in piezo pickup and to a MEMS microphone mounted near the guitar. The guitar used was a Fender Stratocaster acoustic with a built-in Fender pickup. An analog output MEMS accelerometer was mounted on a lightweight flex circuit (Kapton® with etched traces) and attached to the guitar body using beeswax at the bridge location, as shown in Figure 7. The X-axis of the accelerometer was oriented along the axis of the strings, the Y-axis was perpendicular to the strings, and the Z-axis was perpendicular to the surface of the guitar. A MEMS microphone with a flat frequency response out to 15 kHz was mounted 3" from the strings for use as a reference.

Figure 7. Accelerometer mounted on Fender Stratocaster acoustic guitar.

A short sound segment was recorded using the accelerometer, the built-in piezo pickup, and the MEMS microphone. The time domain waveforms for each transducer are shown in Figure 8. No postprocessing was done on any of the audio clips.

Figure 8. Time domain waveforms using different transducers.

Figure 9 shows an FFT-based spectrum of the piezo pickup measured at one of the peaks in the time domain waveform. This spectrum shows a response with a strong bass component. Indeed, the actual audio file sounded excessively full, with a lot of bass response. This sounds pleasing (depending on your taste), as the cavity resonance creates a fuller bass sound than that heard when listening to the instrument directly.

Figure 9. Spectrum of piezo pickup.

The MEMS microphone output is very flat and reproduces the sound of the instrument very well. It sounds very natural, well balanced, and true to life. The FFT-based spectrum measured at the same point in time as the piezo pickup is shown in Figure 10(a). The frequency response of the MEMS microphone is shown in Figure 10(b) for reference.

Figure 10(a). Spectrum of MEMS microphone.

Figure 10(b). Frequency response of MEMS microphone.

The output from the MEMS accelerometer is very interesting. The immediate weak points are that the noise floor was too high and audible at the beginning and end of the track, and that the bandwidth of the Z-axis was clearly limited to lower frequencies. The sound reproduction from each axis was noticeably different.

The X- and Y-axes sounded bright and articulate and had clearly discernible differences in tonality. As expected, the Z-axis obviously sounded bass dominated. Figure 11 shows the X-axis spectrum (a), the Y-axis spectrum (b), and the Z-axis spectrum (c).

Figure 11(a). Spectrum of X-axis.

Figure 11(b). Spectrum of Y-axis.

Figure 11(c). Spectrum of Z-axis.

The X-, Y-, and Z axes mixed together produced a fair representation of the instrument with some brightness. By adjusting the mix, a variation in tonal balance can be achieved with natural sound reproduction. The extended upper harmonics are still missing due to the bandwidth limitation of the current accelerometers, but the sound reproduction was still surprisingly true.

Conclusion

Low-g MEMS accelerometers do not suffer from traditional feedback problems and demonstrate clear potential as high-quality acoustic pickups for musical instruments. A 3-axis accelerometer mounted on a Fender Stratocaster acoustic guitar achieved promising sound reproduction. The three axes have different tonal characteristics related to the vibration modes of the instrument in the different directions of the body. The three output channels can be mixed to generate realistic sound reproduction. In addition, these channels can be mixed in different ways resulting in creative tonal effects.

While the performance of the accelerometer in this experiment is very promising, there are a few drawbacks. The noise floor of the sensor is audible; a problem that can be minimized using noise gating or other techniques, but the ideal sensor will have a noise floor comparable to conventional microphones. The high frequency response of the sensor needs to be extended, ideally up to 20 kHz to capture the full tonal range of the instrument.

MEMS accelerometer technology has clear potential for acoustic pickup applications in musical instruments, especially in live performances where acoustic feedback could be a problem. A very small, low-power MEMS

device can be mounted unobtrusively anywhere on the instrument without affecting its natural vibration characteristics. In fact, multiple sensors can be mounted at different points around the instrument to provide the sound engineer additional flexibility to reproduce the natural character of the instrument without fear of acoustic feedback in live sound application—one step closer to “Sonic Nirvana.”

References

- 1 www.analog.com/en/mems-and-sensors/products/index.html.
- 2 www.analog.com/en/mems-and-sensors/imems-accelerometers/products/index.html.
- 3 www.analog.com/en/mems-and-sensors/imems-microphone/products/index.html.
- 4 Goodenough, F. “Airbags Boom When IC Accelerometer Sees 50 G.” *Electronic Design*. August 8 (1991).
- 5 Rai-Choudhury, P. “MEMS and MOEMS Technology and Applications.” SPIE Press (2000).
- 6 Hopkin, B. “Getting a Bigger Sound: Pickups and Microphones for Your Musical Instrument.” Sharp Press (2002).
- 7 www.analog.com/en/mems-and-sensors/imems-accelerometers/adx1001/products/product.html.
- 8 www.analog.com/en/mems-and-sensors/imems-accelerometers/adx1330/products/product.html.
- 9 Olsen, H. “A History of High Quality Studio Microphones.” *J. of Audio Engineering Society*, 24. December 1976.

10Fontenot, R. "I Feel Fine: The History of this Classic Beatles Song." About.com.

11Freed, A. and O. Isvan. "Musical Applications of New, Multi-Axis Guitar String Sensors." International Computer Music Conference. pp 543 - 546 (2000).

12Olsen, H. "Acoustical Engineering." Professional Audio Journals Inc. (1991).

Authors

Rob O' Reilly [rob.oreilly@analog.com], who began his career at ADI in 1993, is responsible for future business and product development in the Micromachined Products Division. Rob formerly led the Advanced Test, Test, Trim/Probe, and Characterization groups, and, over the past 15 years, has played a key role in the development of iMEMS® accelerometer and gyroscope test, reliability, and characterization processes.

(return to top)

Alex Khenkin [alex.khenkin@analog.com] is a senior acoustics engineer at ADI. With over a decade of experience in microphone research and design at Earthworks, Inc., he has worked extensively on extending the frequency response and dynamic range of microphones, paying special attention to their time-domain characteristics. Alex received his master's degree in applied acoustics at Moscow State Institute of Radio-Engineering, Electronics, and Automation (MIREA). In his free time, he enjoys playing classical guitar.

(return to top)

Kieran Harney [kieran.harney@analog.com], a product line manager in the Micromachined Products Division, is focused on the development of new MEMS technologies. He joined ADI 22 years ago as a semiconductor packaging engineer in Limerick, Ireland; served as assembly and test manager in the Philippines from 1994 to

1997; and assumed responsibility for assembly, test, and advanced package development in Cambridge, MA, in 1997. Kieran received a manufacturing engineering degree in 1983 and an MBA in 1993, both from the University of Limerick, Ireland.

(return to top)

Analog Front End for 3G Femto Base Stations Bring Wireless Connectivity Home

By Thomas Cameron and Peadar Forbes

INTRODUCTION

Imagine a device that can provide high-quality cellular phone reception within your home, allowing you and your family unlimited voice and data usage for a low monthly fee. A femto base station, usually referred to as a femtocell, provides all that and more. This small wireless device, which improves local wireless coverage when placed in a home or office, is poised to dramatically change the wireless infrastructure landscape.

Figure 1 illustrates the femtocell concept. While traditional base stations provide wide area coverage, a femtocell provides wireless coverage in a small area such as a residence. The femtocell routes mobile traffic to the network through the user's broadband Internet connection, thus offloading traffic from the radio network. The femtocell improves the capacity of the network, while reducing backhaul, power, and maintenance costs for the operator. It also enables operators to compete for services in homes that have limited signal coverage. In exchange for a subsidized femtocell, the operator adds an additional fee to the customer's monthly cellular plan. When in the femtocell zone, all mobile usage would be covered under the home billing plan, allowing unlimited voice and data usage in the home without incurring large monthly bills. The proximity of the femtocell enables a high quality link, while simultaneously reducing handset battery usage. The femtocell overcomes the limitation of 3G signals from the base station to penetrate walls, enabling high-speed access to mobile data services such as browsing the Internet, downloading music, and streaming video on the handset.

Figure 1. Femto base station compared to macro base station.

The femtocell, similar to a Wi-Fi router, is based on proven wireless infrastructure standards (UMTS, CDMA). Compatible with emerging standards, it provides an efficient, robust wireless link using operator-owned spectrum. Compatibility with existing handsets makes the connection transparent to the user. Unlike a macrocell network, which aggregates tens or hundreds of base stations onto the core network, a femtocell gateway must manage

thousands or even millions of femtocell nodes.

Femtocells, which must provide the quality of service (QoS) expected from a base station at a cost similar to a handset, present unique challenges to the radio designer. The femtocell must provide both high-quality voice service and high-speed mobile data services (EVDO and HSPA) at a fraction of the cost of a macronode. In order to meet these challenges, the femtocell design must take advantage of low-cost manufacturing techniques and highly integrated circuits that minimize calibration and test time. Femtocells reside in the home, so they must be small, low cost, and user installed. Transmitting at low power—on the order of 100 mW—femtocells must be aware of the wireless environment to mitigate interference and meet regulatory requirements. 3G femtocells must monitor UMTS channels to detect base stations in the vicinity, as well as GSM channels to establish cells that might be appropriate for handover when a user leaves the femtocell zone.

The femtocell can be viewed as two distinct functions: the analog front end and the baseband processor. The front end, which is the topic of this article, converts the digital data stream into an RF signal in the transmit circuit, and vice versa in the receive chain. The front-end design entails trade-offs between integration and performance. Although discrete solutions can be tailored to provide the best performance, the cost would be prohibitive for a femtocell design. Conversely, a fully integrated solution may provide the lowest cost, but the performance may not be sufficient.

Figure 2. Femtocell analog front-end implementation based on ADI chipset.

Figure 2 illustrates a high-level block diagram of a femtocell designed to support local base station operation in UMTS band 1 as well as monitor signals in the 850-MHz, 900-MHz, 1800-MHz, 1900-MHz, and 2100-MHz bands. Together, the AD98631 mixed-signal front-end (MxFE®) baseband transceiver, ADF4602-12 integrated radio transceiver, ADL55423 and ADL53204 linear amplifiers, switches, filters, and other associated support circuitry

form a compact, high-performance front end for the femtocell. A detailed description of the highlighted blocks follows.

On the transmit side, the digital baseband feeds a 12-bit parallel data stream to the AD9863, which converts it to an analog I/Q baseband signal. The baseband signal is converted to RF by the ADF4602-1, amplified by the ADL5542 and ADL5320 gain stages, and sent to a duplexer. A power detector monitors the RF output. A single-pole, six-throw (SP6T) switch selects which transmit or receive monitoring chain is connected to the single antenna. This signal chain provides 13 dBm output power at the RF output connector, while meeting transmit ACLR specifications as defined in 3GPP standard TS25.104.

The receive chain includes surface acoustic wave (SAW) filters and SPDT switches for monitoring the main path. The matching blocks consist of a simple series/shunt inductor for each receive port. The ADF4602-1 has three receiver input pins: one for band 1, and one each for high- and low band monitoring functions. The band-1 receive function may be switched between 1960 MHz to receive the uplink signal and 2140 MHz to monitor the downlink frequency. The ADF4602-1 downconverts and filters the selected RF signal to a baseband I/Q signal. The baseband signal is sampled by the dual ADCs in the AD9863 and converted to dual 12-bit parallel bit streams for the digital baseband.

This functional partition provides the designer with flexibility, ensures high performance in the signal chain, and allows the data converter's speed and resolution to be chosen to fit the application's requirements. The ADI solution enables the designer to combine the analog front end with a commercially available baseband function, accelerating time to market of the femtocell design, while maintaining the benefit of future integration of ADI technology as the femtocell market matures.

ADF4602-1 Integrated Radio Transceiver

The ADF4602-1, shown in Figure 3, is a 3G transceiver offering unparalleled integration and a feature set well-suited to high-performance 3G femtocells. The receiver, based on the direct-conversion architecture, is the ideal choice for highly integrated wideband CDMA (W-CDMA) receivers, reducing the bill of materials (BOM) by fully integrating all interstage filters. The receive baseband filters offer selectable bandwidth, enabling reception of both W-CDMA and GSM-EDGE radio signals. The selectable bandwidth, coupled with the multiband LNA input structure, allow GSM/EDGE signals to be monitored as part of a UMTS home base station.

The ADF4602-1 contains two fully integrated programmable frequency synthesizers for generation of transmit and receive local oscillator (LO) signals. The design uses a fractional-N architecture for low noise and fast lock-time. All necessary components, including loop filters, VCOs, and tank components, are fully integrated for both transmit and receive synthesizers. The VCOs run at twice the high-band frequency and four times the low-band frequency, minimizing VCO leakage power at the wanted frequency and the tuning range requirements of the VCO. The VCOs use a multiband structure to cover the wide operating frequency range. The design incorporates both frequency- and amplitude calibration to ensure that the oscillator is always operating at optimum performance. The fully self-contained calibrations, which occur during the 200- μ s PLL lock time, require no user inputs. The on-chip VCO outputs are fed to tuned buffer stages and then to the quadrature-generation circuitry. The tuned buffers ensure that minimum current and LO-related noise are generated in the VCO transport. The quadrature generators create the highly accurate phased signals required to drive the modulator and demodulator. Special precautions have been taken to provide the isolation demanded by frequency division duplex (FDD) systems between the transmit and receive chains.

Figure 3. ADF4602-1 block diagram.

The receiver front-end includes three high-performance, single-ended, low-noise amplifiers (LNAs), allowing the device to support tri-band applications. Two are suitable for high-band operation from 1800 MHz to 2170 MHz,

while one is suitable for operation from 824 MHz to 960 MHz. Interstage RF filtering is fully integrated, ensuring that external out-of-band blockers are suitably attenuated prior to the mixer stages. The single-ended 50- Ω input structure eases interfacing and reduces the matching components required for small footprint single-ended duplexers. The excellent device linearity ensures good performance with a large range of SAW- and ceramic filter duplexers.

High linearity demodulator circuits are used to convert the RF signal to baseband in-phase and quadrature components. Two demodulator sections are included: one optimized for the high-band LNA outputs and one optimized for the low band. The high-band- and low-band outputs are combined to drive directly into the first stage of the baseband low-pass filter, which reduces the largest blocking signals prior to baseband amplification. The receiver synthesizer section provides quadrature LO drive to the mixers from the VCO distribution system. A programmable divider allows the same VCO to be used for both high- and low bands. Excellent 90° quadrature phase- and amplitude match are achieved by careful design and layout of the demodulators and VCO distribution circuits.

The baseband section, which includes distributed gain and filtering, is designed to provide a maximum of 54-dB gain with a 60-dB gain-control range. Through careful design, pass-band ripple, group delay, signal loss, and power consumption are kept to a minimum. Filter calibration is performed during the manufacturing process, resulting in a high degree of accuracy and ease of use. Two selectable 7th-order baseband filters are available: one with a 1.92-MHz cutoff for W-CDMA and one with a 100-kHz cutoff for GSM.

In W-CDMA mode, the ADF4602-1 is capable of providing 102-dB gain with a 90-dB gain-control range distributed throughout the receive signal chain. The RF front-end contains 30-dB of control range: 18 dB in the LNA and 12 dB in the mixer transconductance stage. The two baseband active filter stages each provide an 18-dB gain-control range in 6-dB steps. This results in a 36-dB total gain control range in three 12-dB steps. The variable-gain amplifier (VGA) implements a 24-dB gain control range in 1-dB steps. To simplify programming

and ensure optimum receiver performance and dynamic range, simply program the total desired receive gain; the ADF4602-1 decodes the gain setting and automatically distributes the gain between the various blocks.

The transmitter uses an innovative direct-conversion modulator, which achieves high linearity and low noise while eliminating the need for external transmit SAW filters. The differential, dc-coupled baseband interface for I and Q channels supports a wide range of input common-mode voltages (VCM) from 1.05 V to 1.4 V. The maximum allowed signal swing is 550 mV peak, which corresponds to a differential range of 1.1 V p-p on either I or Q channels. Prior to the quadrature modulator, the baseband inputs' signals pass through a 2nd-order Butterworth filter with a cutoff frequency of 4 MHz to suppress out-of-band spurs. Calibration techniques maintain accurate I/Q balance and phase across frequency and environmental conditions, ensuring that 3GPP carrier leakage, EVM, and ACLR requirements are met with good margin under all conditions. The ADF4602-1 achieves a -163 -dBm/Hz broadband noise floor at a 190 MHz offset and -8 -dBm output power, while meeting TS25.104 requirements for EVM and ACLR. The output is matched to 50Ω to enable a simple connection to the power amplifier.

AD9863 Mixed-Signal Front-End Baseband Transceiver

The AD9863, a member of the MxFE family of integrated converters for the communications market, is ideally suited for low-cost, high-performance femtocell applications. It integrates dual 12-bit analog-to-digital converters and dual 12-bit TxDAC® digital-to-analog converters. The ADCs are optimized for sampling at 50 MSPS or less. The DACs, which operate at speeds up to 200 MHz, include a bypassable $2\times$ or $4\times$ interpolation filter. Packaged in a 64-lead LFCSP package, the AD9863 is only $9 \text{ mm} \times 9 \text{ mm} \times 0.9 \text{ mm}$. The AD9863 is highlighted here, but other members of the MxFE family (AD9860, AD9861, and AD9862) offer the designer flexibility in choice of performance and auxiliary converters for control circuits.

Figure 4. AD9863 MxFE block diagram.

A flexible, bidirectional 24-bit I/O bus accommodates a variety of commercially available baseband ASICs or DSPs. In half-duplex systems, the interface supports 24-bit parallel transfers or 12-bit interleaved transfers. In full-duplex systems, the interface supports a 12-bit interleaved ADC bus and a 12-bit interleaved DAC bus. The flexible I/O bus reduces pin count and package size. For frequency division duplex (FDD) W-CDMA, the AD9863 operates transmit and receive channels simultaneously. This requires the use of full duplex mode—one 12-bit interleaved Rx data bus and one 12-bit interleaved Tx data bus.

The DAC core converts the 12-bit data into two complementary differential current outputs, providing them to the ADF4602-1 using a resistor network, as shown in Figure 5. RDC is set to 120 Ω for a 1.2-V common-mode voltage, and RL is set to 63 Ω for a 1-V p-p differential input swing.

Figure 5. Simple interface between the AD9863 and ADF4602-1.

The DACs contain programmable fine-gain- and dc-offset controls that can be used to compensate for mismatches between I and Q channels to suppress LO feedthrough and improve EVM performance. The 10-bit dc-offset controls can be used independently to provide up to $\pm 12\%$ of offset to either differential pin, thus allowing calibration of any system offsets.

The ADC input consists of a 2 k Ω differential input resistance and a switched capacitor circuit. The input can be self biased to midsupply, or it can be programmed to accept an external dc bias. It is thus recommended that the ADF4602-1 receive baseband outputs be connected directly to the AD9863 ADC inputs. The ADC input full-scale level is 2 V p-p differential.

Clock Solution for the Femtocell

The femtocell requires a very accurate reference clock— ± 0.1 ppm—in order to meet 3GPP specifications.

Methods for implementing this very fine clock control are outside the scope of this article—but a number of possibilities exist, including GSM macrocell synchronization via the monitoring receivers, GPS synchronization, and IEEE 1588 precision timing protocol. In some instances, a combination of the above methods may be implemented by femtocell vendors. Ultimately, the reference timing control circuitry will regulate the reference frequency source. On the ADI evaluation board,⁵ this 26-MHz VCTCXO is used as the reference to the ADF4602-1. A delay-locked loop (DLL) generates 19.2 MHz, which is a multiple of the 3.84-MHz W-CDMA chip clock. This 19.2-MHz clock is used as the clock input for the AD9863.

The AD9863 has a versatile clocking configuration with many variables. The ADC clock rate, DAC clock rate, PLL, and interpolator settings are software controllable, allowing optimization of power vs. performance to suit the requirements. In the recommended configuration, the PLL multiplier is set to $2\times$, giving a PLL output frequency of 38.4 MHz. The ADC is clocked at half this frequency. On the transmit side, the 38.4-MHz PLL output is used to clock the DAC. Transmit interpolation is set to $2\times$ in order to suppress DAC images. Other combinations of clock frequencies are also possible. The AD9863 data sheet provides a complete description of the operating modes. Using the above clock scheme, the femtocell does not require any discrete frequency conversion PLLs, as are often found in macrocell base stations. All frequency conversion is integrated, helping the femtocell to meet the price point demanded by the market.

RF Amplifiers

The amplifiers chosen for the RF power stage are low-cost, high-performance, broadband linear amplifiers fabricated on an InGaP process. They linearly amplify the output of the ADF4602-1 and compensate for losses in the RF duplexer and switching networks. The ADL5542 contains internal biasing and matching; the ADL5320 requires external matching, and is packaged in an industry-standard plastic SOT-23 package. Both amplifiers run directly off a 5-V rail, so no external bias circuitry is required. Key specifications for the amplifiers are shown in Table 1. Proprietary techniques applied to the design of the ADI RF amplifiers provide exceptional linearity vs.

supply current.

Table 1. Key Specifications for the ADF5542 and ADL5320 (@ 2 GHz)

Specification	ADL5542	ADL5320
Gain	19 dB	13.2 dB
P1dB	18.9 dB	25.7 dBm
Output IP3	37 dBm	42 dBm
Noise Figure	3.1 dB	4.4 dB
Supply Current (5 V Supply)	97 mA	104 mA

Transmit Output Power and Interference Mitigation

To mitigate interference, the femtocell must set its output power flexibly and intelligently to account for deployments where multiple femtocells operating on the same frequency are located in close proximity to each other (e.g., in an apartment complex). Here, each femtocell will need to transmit at lower output powers to avoid same-frequency interference. Also, the femtocell cannot cause interference to geographically neighboring macrocell base stations operating on the adjacent channel, as this would create dead spots for nearby mobile phones connected to the macrocell network. The femtocell will thus have an adjacent channel protection requirement, forcing it to measure the power in the adjacent downlink channel and set its own power according to a predetermined formula so as not to obstruct the macrocell signal.

To allow the femtocell to meet the price point required and for ease of customer installation, these interference mitigation techniques must be automatic and must not require input from a trained field technician or the home user. This process should be automatically initiated when the box is first turned on by the user, and updated at regular intervals thereafter. Together, the band 1 monitoring receiver on the ADI design and the large transmit dynamic range available on the ADF4602-1 allow the femtocell vendor to implement these interference mitigation

techniques automatically without external input. The monitoring receiver allows the power in the adjacent channel to be measured accurately and the output power to be adjusted accordingly. About 30 dB of total transmit power dynamic range will be required.

Radio Performance Measurements

To evaluate the transceiver chipset against the TS25.104 radio systems specifications, the transceiver lineup described above has been incorporated into an evaluation board design. The evaluation platform, shown in Figure 6, enables the independent testing of transmit and receive chains, as well as individual component testing. The evaluation board includes the functionality of the block diagram in Figure 1 as well as power conditioning. The radio portion, including ADF4602, AD9863, ADL5542, ADL5320, VCTCXO, and all associated front-end switches and filters, occupies a 1" × 2" space on the board. Note that this board has not been optimized for space savings as it is provisioned for testing purposes, but a more compact design can be achieved for production. Some of the key test results against the TS25.104 specifications are included below to illustrate the performance of the ADI chipset on the evaluation board.

Figure 6. ADF4602-1/AD9863 evaluation board.

Figure 7 shows the band 1 receiver sensitivity measurement. Receiver sensitivity is a measure of how well the receiver can detect a low-level signal, and is an indicator of the noise figure of the receiver. In this measurement, a 12.2 kHz reference is used. The ADF4602-1 gain is set to 80 dB. The receiver sensitivity exceeds the TS25.104 specification by 6 dB or more across the band.

Figure 7. Band 1 receiver sensitivity.

Another key specification for the receiver is the performance under blocking conditions. The blocking tests

simulate the ability to receive the wanted signal in the presence of large unwanted signals in adjacent channels. The UL 12.2-kHz reference signal is set to -101 dBm, and blocking signals are injected until a BER of 10^{-3} is measured. As shown in Table 2, the ADF4602-1 exceeds the TS25.104 with some margin in all three cases.

Table 2. Summary of Receiver Blocking Testing vs. TS25.104 Specifications

Receiver Blocking Specifications	TS25.104 Specification	Limit	ADF4602 Evaluation Board Test Results
Adjacent Channel Selectivity	-38 dBm	-31 dBm	(7 dB margin)
10 MHz WCDMA Blocker	-30 dBm	-21 dBm	(9 dB margin)
20 MHz Out of Band CW Blocker (1900 MHz)	-15 dBm	-11 dBm	(4 dB margin)

Key indications of transmit chain quality are adjacent channel leakage ratio (ACLR) and error vector magnitude (EVM). In both cases these tests are key indicators of the linearity of the combined transmit chain. Table 3 compiles the measurements taken on the ADI evaluation board compared to the TS25.104 specifications. It also includes peak code domain error, an EVM measurement that ensures even spreading of errors over the code domain.ii In all cases, the ADF4602-1 evaluation board exceeds the TS25.104 specifications with margin. A plot of the output spectrum used in the ACLR measurements is shown in Figure 8.

Table 3. Summary of Transmitter Testing vs. TS25.104 Specifications

Transmitter Specifications	TS25.104 Specification	Limit	ADF4602 Evaluation Board Test Results
Error Vector Magnitude (EVM)	<12%	4%	
Peak Code Domain Error (PkCDE)	< -33 dB	-46 dB	
Adjacent Channel(5 MHz) ACLR	< -45 dB	-49 dB	
Alternate Channel (10 MHz) ACLR	< -50 dBm	-72 dB	

Figure 8. ACLR measurement for W-CDMA band 1 signal with 13 dBm output power.

Figure 9 shows a transmit EVM plot for a typical femtocell configuration involving two HSDPA channels and a number of voice/data channels. The composite EVM is below 4%. Evaluation of the circuit has shown that the EVM is dominated by the LO leakage introduced by the I/Q offsets voltage at the input of the modulator—a feature of direct-conversion transmitters. As mentioned above, these offsets may be calibrated out by using the AD9863 dc-offset controls. A complete description of this calibration method is provided in the application note.⁶

Figure 9. EVM measurement for a typical femtocell configuration.

CONCLUSION

The emerging femtocell application presents a unique challenge to the radio designer to minimize cost while maintaining base-station performance. The ADI 3G femtocell chipset comprised of the ADF4602-1 integrated radio transceiver, AD9863 MxFE baseband transceiver, and ADL5542 and ADL5320 RF amplifiers enables the femtocell designer to meet the TS25.104 specifications in a compact form factor.

REFERENCES

- 1 www.analog.com/AD9863
 - 2 www.analog.com/en/rfif-components/rfif-transceivers/adf4602-1/products/product.html
 - 3 www.analog.com/ADL5542
 - 4 www.analog.com/ADL5320
 - 5 ADF4602 and AD9863 Femtocell Evaluation Board, EVAL-ADF4602EB1Z Data Sheet.
 - 6 Transmit EVM Calibration on the ADF4602 W-CDMA Transceiver. AN-xxx
- iTSG R4#48 – TSG-RAN Working Group 4 (Radio) meeting #48 October 2008.
iiTSG R4#8 (99)705 – TSG-RAN Working Group 4 (Radio) meeting #8. October 1999.

THE AUTHORS

Thomas Cameron (thomas.cameron@analog.com) joined ADI in 2006. Currently a technical business manager concentrating on the wireless infrastructure vertical market, he began his career in 1986 in the Radio Networks Division at Bell Northern Research (now Nortel Networks), where he held various positions spanning research, design, and management of technology, devices, and subsystems for telecom networks. In 1999, he joined Sirenza Microdevices, where he advanced to director of marketing for wireless products. In 2004, he moved to WJ Communications, where he held the position of European sales manager. Thomas holds a B.Sc. from Wilfrid Laurier University in Waterloo, Canada; an M.Eng. in Electrical Engineering from Carleton University in Ottawa, Canada; and a Ph.D. in Electrical Engineering from the Georgia Institute of Technology. He holds seven patents and has authored numerous technical papers and articles.

(return to top)

Peadar Forbes (peadar.forbes@analog.com) joined Analog Devices in 2004, following his graduation from University College Cork, Ireland, with a Bachelor of Science in Microelectronic Engineering. He currently works for the RF product line, providing applications support for RF transceivers and PLL products. In his spare time Peadar enjoys music, playing guitar, sports, and travel.

(return to top)

Open a Dialogue

Question the authors. Share information with your colleagues. Leave feedback for the editors.

What did you think of this article? Was it useful, timely, well written?

Would you like to see more articles on this topic?

Please leave your comments at Analog Diablog.

Copyright 1995- 2009 Analog Devices, Inc. All rights reserved.

StatCounter - Free Web Tracker and Counter