The Ensembl Computing Architecture

James A. Cuff,[2,3] Guy M.P. Coates,[1] Tim J.R. Cutts,[1] and Mark Rae[1]

[1]*The Wellcome Trust Sanger Institute, The Wellcome Trust Genome Campus, Hinxton, Cambridge, CB10 1SA, UK;* [2]*The Broad Institute, Cambridge, Massachusetts 02141, USA*

Ensembl is a software project to automatically annotate large eukaryotic genomes and release them freely into the public domain. The project currently automatically annotates 10 complete genomes. This makes very large demands on compute resources, due to the vast number of sequence comparisons that need to be executed. To circumvent the financial outlay often associated with classical supercomputing environments, farms of multiple, lower-cost machines have now become the norm and have been deployed successfully with this project. The architecture and design of farms containing hundreds of compute nodes is complex and nontrivial to implement. This study will define and explain some of the essential elements to consider when designing such systems. Server architecture and network infrastructure are discussed with a particular emphasis on solutions that worked and those that did not (often with fairly spectacular consequences). The aim of the study is to give the reader, who may be implementing a large-scale biocompute project, an insight into some of the pitfalls that may be waiting ahead.

In the beginning, flat files were the mainstay of bioinformatics. Most data could be easily represented as a sequence of ASCII characters. The number of known proteins was small, and genomes were a mere thought for the future. Sequence analysis became much more difficult during and after the advent of rapid DNA sequencing (Sanger and Coulson 1975).

The problem became harder still as researchers delved into the world of sequence comparison (Needleman and Wunsch 1970; Smith and Waterman 1981; Pearson and Lipman 1988; Altschul et al. 1990). It was rapidly discovered that techniques such as dynamic programming would be essential to solve any all-by-all sequence comparison problem. Flat files and single CPU servers were just not going to be enough for the task of whole-genome comparison and annotation.

Ensembl began as a project to automatically annotate vertebrate genomes. This initially involves running many commonly used bioinformatics tools such as gene-prediction algorithms, sequence database searches, and genomic feature-finding programs. Ensembl takes the output features from these tools and combines them to produce genome-wide data sets such as protein-coding genes, genome–genome alignments, synteny maps, and protein family clusters. Details of how this is done are described elsewhere in this issue.

For a genome of the size of human, the number of CPU hours needed to produce this annotation can run into many thousands. Obviously, this is not useful to complete on a single CPU, but the problem can be divided up into many smaller tasks that can be distributed over multiple CPUs. For large numbers of CPUs, this reduces the time taken drastically.

Initially consisting of the human genome, the project has subsequently grown to encompass mouse, rat, zebrafish, and fugu, to name but four. Accordingly, the compute requirement for the project expanded. Starting with one 466-MHz Alpha DS10, and 200 GB of storage, to what is now a sizable enterprise class-computing infrastructure, comprising >1200 CPUS and >20 TB of online storage (Fig. 1). On the basis of these figures, genome analysis requirements exceed expectations of CPU computational scaling (Moore 1965).

[3]Corresponding author.
E-MAIL jcuff@broad.mit.edu.; FAX (617) 258-0903.
Article and publication are at http://www.genome.org/cgi/doi/10.1101/gr.1866304.

METHODS

Design

The Problem

As stated, Ensembl needs to compute sequence similarities and execute pattern discovery algorithms to produce gene structures. The data is also denormalized by EnsMart to allow rapid search and queries to take place. The construction of the EnsMart database is a sizable task that requires access to all current Ensembl databases. The current sequence databases that are required for an Ensembl gene build are >50 GB in size, with a final build database size that is >200 GB.

Servers and architecture design for this project include commodity hardware with small memory configurations, and small disk units, up to large SMP servers with fast RAID storage to hold the core databases.

Fortunately, the algorithms that are executed fall mainly into the class of problems that, in computer science, are termed as 'embarrassingly parallel' (Fox et al. 1994). Classical parallel processes need to communicate and update information between threads or instances of the parallel process. An embarrassingly parallel application does not require interprocess communication. As such, each thread or section of the problem is independent and can be executed without reference to any of the other processes. If this were not the case, farm architectures would not be applicable for this task.

Hardware and CPUs

The heart of sequence comparison is an inherently integer-based problem. CPUs with good integer performance and sufficient level-two caches perform exceptionally well with dynamic programming tasks. Certain sequence comparison tasks, such as BLAST (Altschul et al. 1990) also demand considerable input/output (I/O) subsystems. They read large files and write considerable output. For these types of algorithms, localized storage is essential to bind the data files as tightly as possible to the core CPU.

To circumvent the problem of data localization, all data files (e.g., binary database indices and raw sequence files) are distributed to each remote node's local storage device. Because they are local to the machine, no NFS requests need to be made, as the remote farm node disk already contains the binary and data files.

However, there are even issues with this approach, as single disk speed and overall disk performance often becomes the limiting factor when reading large data sets. Fortunately, many op-

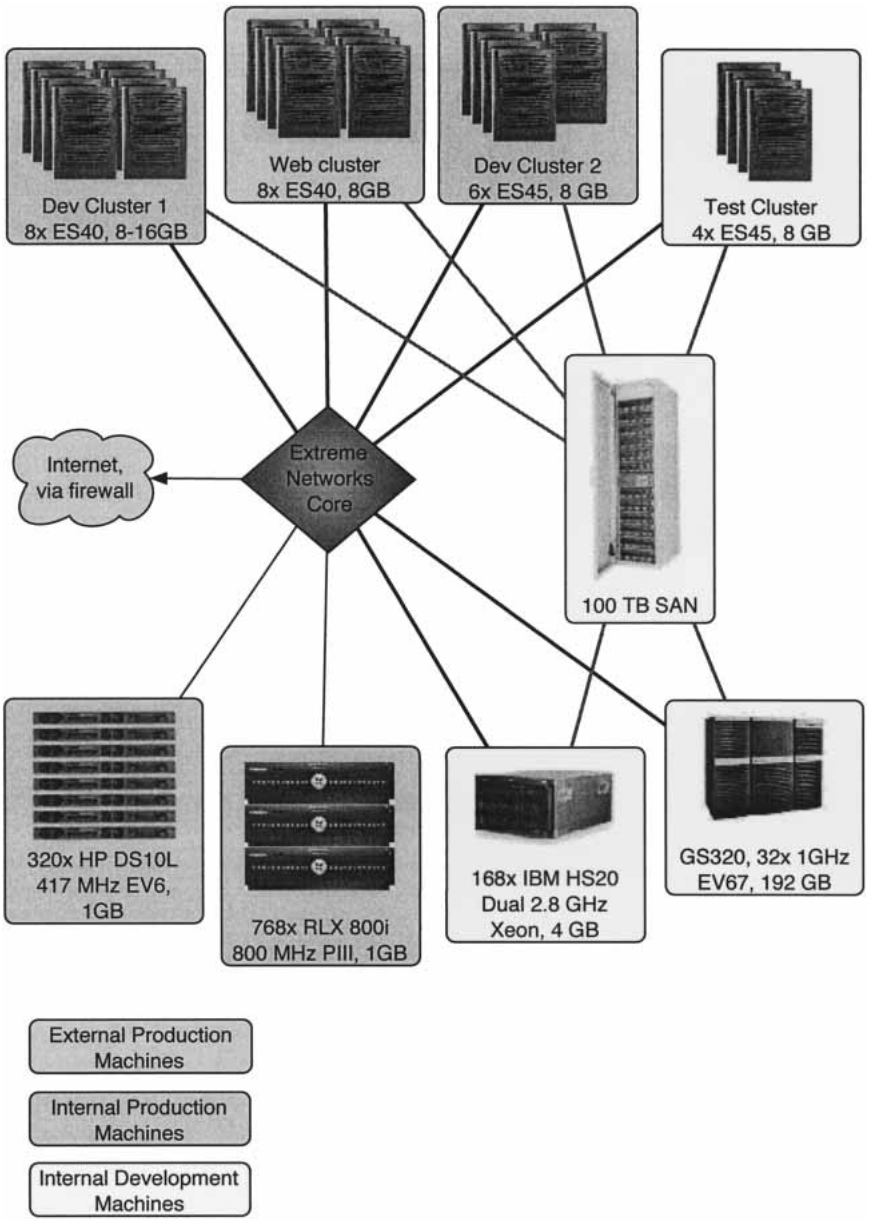

Figure 1 Ensembl Computing infrastructure.

erating systems allow one to construct either software and/or hardware RAID volumes, which enable multiple disks to be bonded together to aggregate the I/O and improve performance.

When the server disk I/O subsystem becomes rate limiting, further steps must be taken. For example, IDE disk drives in certain blade servers have a maximum throughput of 20 MB/sec. To further increase I/O performance, multiple fiber-host bus adapters (HBAs) can be used. Attaching multiple HBAs to a server enables rapid file access for both read and write. This type of connection is better suited to file servers and head nodes than for remote client machines, due to the increased cost. Head nodes in this case would be the ES40/45 clusters as shown below (Distributed Resource Management). However, blade vendors now enable fiber-attached storage direct to the farm node, which gives even further flexibility.

The concept of head nodes is vital in this type of cluster configuration. A head node is in essence a larger (often multi-CPU) server, with a large amount of core storage behind it. These servers run the master batch daemons for queue management, serve core NFS directories to the farm, and also run the relational database engines to supply the farm with data.

Databases

From the early days of the Ensembl project, it was clear that relational databases (RDBs) would be an essential requirement for the ultimate success of the project. Ensembl was founded on open source principles; as such, all code must be freely distributable to all. In light of this, MySQL was chosen as the core database engine.

The RDB forms the core anchor point for the design and construction of the systems architecture. In the beginning, there was one core MySQL database, with 40 nodes working as slave compute devices. There are now 16+ database instances with >400 databases within them. Total storage requirement today for MySQL alone exceeds 4 TB.

The gene-build process involves a combination of multiple SQL select statements along with writes access to specific tables to store the results of a given analysis task. The database architecture must be designed such that each task is not rate limiting.

Figure 2 shows an outline of how the storage is used in an eight-node cluster. Here, rapid I/O and good CPU performance is vital. Accordingly, enterprise class servers with fiber-channel storage devices and low-latency interconnects were deployed. For Ensembl, Compaq/HP ES40 and ES45 servers with memory-channel interconnect, and HSG/HSV RAID storage arrays are deployed. Up to eight-way speed up for SQL select statements can be achieved by deploying a read-only database as depicted in Figure 2.

The SQL select statements are sent to the cluster alias, which transparently balances the MySQL database activity across eight machines holding read-only copies of the data.

This read-only replication is possible because the cluster has both a shared file system and a cluster alias that can route network traffic to any node in the cluster via a round-robin mechanism. All nodes in the cluster are able to see the same storage, as each separate MySQL instance is able to see the same data files.

One word of caution with this approach, because there is no locking in the database, only read access is permitted. Actions that involve changing or writing new data into the database cannot use the alias and must, instead, be passed to a specific database server. This read-only replication mechanism is also deployed in the Ensembl Web site to improve throughput, where the Web site requests are mainly read only.

Distributed Resource Management

Some kind of distributed resource manager (DRM), often called batch-queuing system, is essential to enable optimal use of any

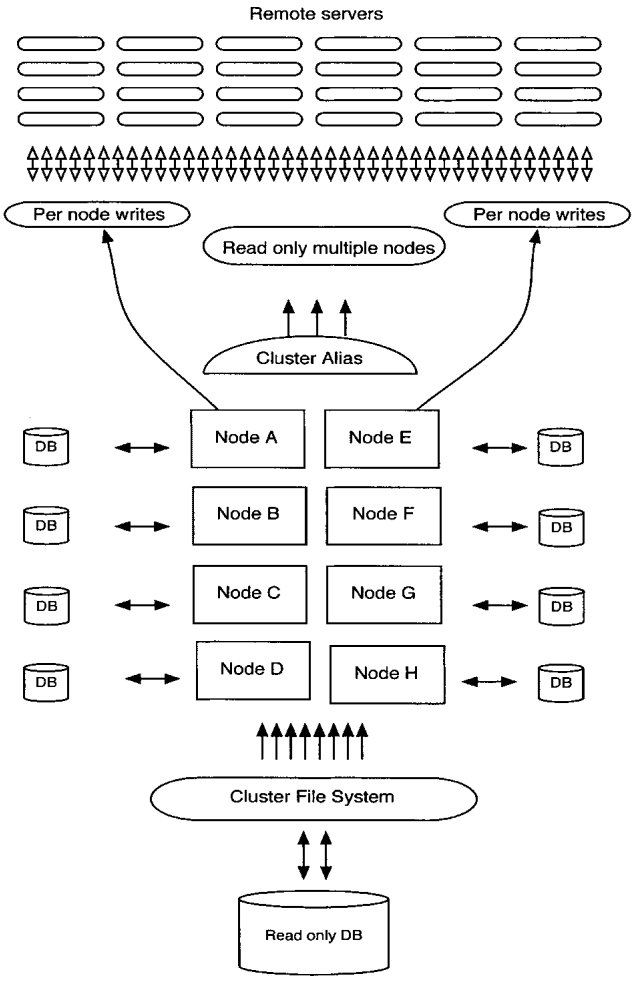

Figure 2 Database layout on an 8-node cluster. To enable distribution of computational load, remote devices can access the cluster alias to read from the replicated database. If write access is required, individual nodes must be specified. The cluster alias access is very efficient for large SQL select statements where speed is required.

farm setup. To work with clusters without a low latency interconnect, work load must be able to be parallelized simply by breaking the job down into simple units. However, to achieve this level of parallelism, one requires vast numbers of individual jobs. The queuing system is essential to handle these multiple jobs. In the case of Ensembl, the number of jobs can approach 300,000 in total, with up to 30,000 executing concurrently.

```
  PID USERNAME SIZE   RES STATE   TIME   CPU  COMMAND

  8241 james    83M 4956K WAIT    0:00  4.80% <wublastn>



Jul 28 16:53 vmunix: NFS3 server master not responding still trying

Jul 28 16:58 vmunix: NFS3 server master ok
```

Figure 3 Client code in wait state. Note, the CPU time is pitiful, the process state is in WAIT, as it is waiting for I/O operations on the NFS server. This can be seen from the kernel messages file above, which also shows the server (master) to be unresponsive.

There are many options as follows:

LSF; http://www.platform.com/products/LSF
PBS; http://www.openpbs.org
GridEngine; http://gridengine.sunsource.net
NQS; http://umbc7.umbc.edu/nqs/nqsmain.html,
and even UNIX at and batch commands, to name but a few.

The system deployed for Ensembl used LSF. This was due in part to the requirement for a commercial strength, 24 × 7 support contract; there was also in-house experience of the product; therefore, it was a relatively quick system to deploy. Many vendors offer this level of support and ease of configuration today.

The importance of distributed resource management should never be overlooked when building such clusters. It is a technology that is under constant research and development, and due diligence must be used when deciding which to deploy. The source code for the Ensembl pipeline has been modified to use a variety of queuing systems.

Scaling Issues and Failure Modes

Table 1 outlines some of the significant areas in which large farms will fail. These errors are often not significant and will not be seen where system size is small; they only start to occur when the node count is high. Fortunately, most of the key failure modes do have practical work around that are simple to implement. For example, the e-mail output is easy to resolve by turning off the sendmail daemon. E-mail on a cluster is not required. By reference to Table 1, each section that follows will explain in further detail what the issue is and how it may be best resolved.

NFS

Deploying NFS in clusters effectively is nontrivial. To work well, it involves making sure as much data is as close to the servers as practically possible. Configuration data and small scripts are good examples of items that can be shared successfully via NFS. Data files are a different issue all together. Accordingly, client servers need to be specified to have some form of local disk storage, where both copies of the executables and the query data files can be stored. This reduces NFS overhead dramatically, as the queries all run with locally served data files.

NFS has been a difficult issue since the inception of the cluster. Many hardware and software technologies were deployed to improve this situation.

- Use trunking to bond multiple network interfaces to double performance.
- Use multiple NFS servers, for example, one for binaries and one for perl modules.
- Upgrade network infrastructure; initial servers were on 155MB ATM, gigabit networking increased performance dramatically. This provided the best increase in performance of all methods.
- Use Jumbo frames on the gigabit network so that the maximum transfer unit is increased. In our experience, this made little difference, as the host CPU was rated at 2.8 GHz, and was able to assemble the smaller 1500-byte packets with little loss in performance.
- Cluster NFS—use Tru64 cluster aliases to balance the load over multiple members in the cluster. This had the same effect as using multiple servers.

The following describes a scenario in which wuBLASTN is executed against an 800-MB subsection of the human EST

Table 1. Issues Where Farms Fail

Email	Sending email from x,000 nodes at the same time with the results of output, e.g., LSF's default behavior sends stdout and error via email. Often results in a crashed mailserver.
NFS	Remote file read access over the network from multiple nodes.
NFSKiller	As for nfs, but with multiple write access all at the same time.
PreExec	Writing a pre-execution wrapper to test a failure condition prior to the job running, that then fails to run, or exits with a non-zero exit status. e.g., coding error can cause this.
RubberJob	Jobs bouncing in the queues due to some failure state, e.g., missing nfs intercept point, coding error. Often induced by a PreExec failure–as above.
Typo	Often the biggest killer of farm servers, e.g., /daata/blast not /data/blast, jobs can become Rubber, and proceed to bounce in and out of the queuing system on to modes and then fail.
RawOut	Writing raw blast/exonerate/other output without any data reduction, e.g., MSPCrunch/grep, etc.
BigLog	Excessive logging that may generate more than 1 GB of stderr output data, this is often written back to the NFS server.
JobSize	Jobs that run for less than 1 sec. Also jobs that run for 6 mo.
SwapKiller	Jobs that end up allocating too much memory, or jobs that grow and difficulty predicting usage patterns, e.g., exonerate FSM generation.
MasterKiller	Job submission, dispatch rate, and queue size are sufficiently high that the dispatch code becomes CPU bound and fails to run new jobs.
NameService	NIS or DNS servers become overloaded due to many gethostsbyname, getuid, getgroup requests.
NetFill	Backbone network becomes saturated with I/O requests, e.g., heavy NFS or DBI loading.
DeadDisk	Storage failure on remote node. When large numbers of spindles are considered, this becomes a significant factor. Jobs can arrive at the remote node and find that the storage has failed.
RDBKiller	Head database nodes become saturated with long running requests, or too many concurrent connections. RDB is no longer able to actively serve results for new SQL statements.

There are a number of bottlenecks that can appear in farm environments; the table above is a list of some of the significant ones. If you are particularly unlucky, you will see all of them at the same time.

database. There are only five short query ESTs used in the search; this is not a large-scale comparison. The clients are DS10L servers, with two bonded 100-Mbit Ethernet devices to provide 20 MB/sec bandwidth to the server. Storage devices are simple IDE disk devices. The NFS master is a 4 CPU AlphaServer ES45, with Gigabit networking with fast Fibre Channel RAID storage. The results are shown in Table 2.

However, attempting to contact the master server during the 200-node access via NFS results in no response. We need to find out why. Looking at the client side process list shows poor CPU response, (see Fig. 3). Looking at the process list on the master shows heavy I/O from the kernel thread (see Fig. 4).

Note, from Figure 4 one can see that the second process in the list also now has no user id associated with it, just a number. We have essentially destroyed the YP server; we cannot resolve user id data any longer. Albeit that the server has a 1-GB/sec network interface, it can no longer serve requests; the link is saturated. This server may as well be off line, as it also serves users' home directories; therefore, no one can login to this server. By now, telephones in the systems' support room will be ringing.

Table 2. Execution Time for wublastn Jobs With 5 EST Sequences

Mode	CPU Seconds
One host with local storage:	226 seconds
One host (NFS):	220 seconds
Thirty hosts (NFS):	282 seconds
Two hundred hosts (NFS):	1121 seconds
Two hundred hosts with local storage:	226 seconds

Other Issues

Table 1 also identifies a number of other issues that can kill servers. Other than the user errors, such as Typo, BinLog, and SwapKiller, system errors can be avoided by replicating services. For example, NameService can be avoided by either replicating NIS servers as a series of slaves or using local caching services such as nscd. nscd will cache common name-service requests such as hosts, group, and id requests. This stops the YP server from becoming saturated. Again, this issue only presents itself in big installs, where the number of users and servers are large.

The same database replication approach discussed above (Databases) can also help to resolve RDBKiller issue. Multiple nodes to serve the SQL-select statements reduce the bottlenecks considerably.

Compute Farm and Data Access Architecture

Distribution of the analysis over the computers in a farm can generate large amounts of network traffic. However, for the programs to perform optimally, we need to restrict the network traffic associated with running jobs as much as possible. To do this, a number of programs have been devised over time to push data to remote nodes.

File distribution with rsync/rdist programs is slow when the node count is high. It can take up to 2 d to distribute the entire contents of the sequence databases. A total of 60 GB of data pushed to 1200 nodes is in essence 72 TB of data that must move down the wire. Without careful design, this is a huge data-distribution problem.

We circumvent the data-distribution problem by applying the same embarrassingly parallel techniques that we use for executing code. We split the farm into a series of head nodes, which are each responsible to distribute data to a small number of children. In this way, we obtain data parallelism, and reduce the time to distribute the data to less than 1 d. The issues that increase the time for data copying are annoying issues, such as full disks and failed disks. The farm has >2256 disk drives that are >135 TB in

```
  PID USERNAME    SIZE    RES STATE    TIME    CPU  COMMAND

15284 root         10G   345M run    100.0H 104.9% <kernel idle>

16202 10965      2704K 1687K run     52:07  99.9% <xdelta.osf>
```

Figure 4 Server code in kernel thread state. The kernel idle process here is the NFS kernel thread in the server trying desperately to serve NFS requests.

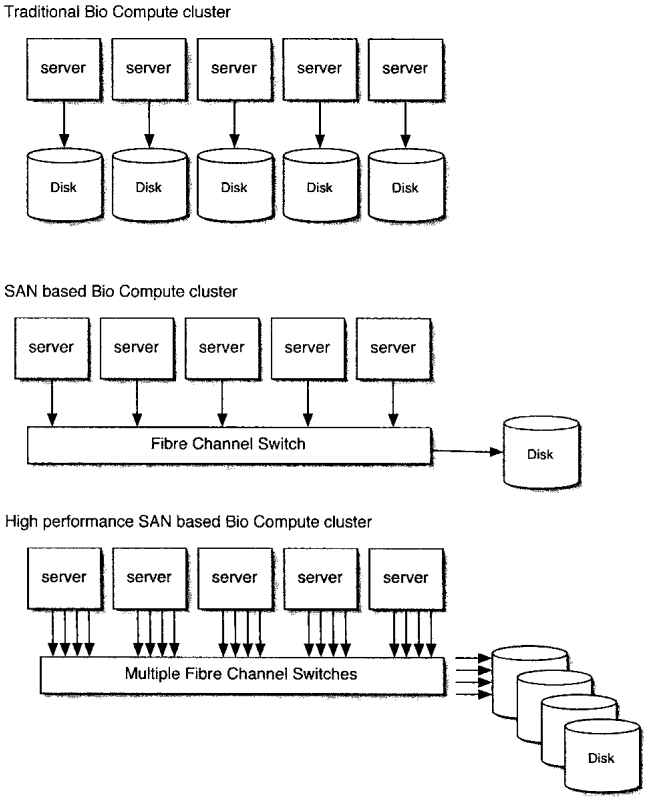

Figure 5 Traditional vs. SAN storage for clusters.

total. Mean time between failure becomes an issue when there are >2000 disk drives to deal with. This accounts for the DeadDisk issue described in Table 1. On the Ensembl cluster, SMART disk failure code was written to automatically close hosts when storage was shown to be in a pre-fail state.

Initial experiments with multicast (push data to all nodes at the same time) failed. Reliable multicast is difficult, as the underlying UDP protocol has no concept of byte order or guaranteed delivery. Recently, however, code based on udpcast (Knaff 2003) have been deployed. Udpcast is a stable multicast code that is able to distribute multigigabyte files at close to wire speed. This code has been recently tested over the entire farm with good results.

Due to the vast number of storage devices deployed, and the fact that they are the only item with moving parts in a server (other than cooling fans), this has been the hardest task to maintain service. It was decided that consolidating this storage was the only way to reduce this overhead.

The latest addition of hardware to the farm uses SAN technology. Because the data analysis problem on the farm is inherently read only, tricks can be deployed where storage is presented to multiple nodes as read-only file systems. Because they are read only, the same device can be presented to multiple operating systems. If this were a read-write volume, the journal logs would corrupt instantly, and the storage would need to be reinitialized.

Figure 5 shows a schematic of how a read-only arrangement works in practice. The bottlenecks tend to move to the switch fabric and the speed of the interswitch links (ISLs). From our tests running four ISLs with two controllers, we can sustain 588 MB/sec peak speed, which is ca 44 MB/sec per node.

DISCUSSION

Conclusions

The most effective method to maintain a highly available compute environment is to keep the components as simple as possible. Only use expensive enterprise class servers where you really need them, for example, database engines.

Localizing data files is essential to reduce internal denial of service due to code design failure or systems failure.

Storage-area network technologies, while currently expensive, are an effective way to tightly couple both storage and servers, and provide exceptional I/O performance.

Read-only problems, such as sequence comparison and SQL selects, can be effectively accommodated by multiple-device presentation to load balanced servers and by cluster-wide file systems.

Looking forward, plans to increase the amount of directly attached storage and to extend this direct attached storage out the farm nodes are underway, as is the provision of cluster-wide file systems to the farm nodes.

ACKNOWLEDGMENTS

Ensembl is funded principally by the Wellcome Trust with additional funding from EMBL and NIH-NIAID. We thank Andy Smith and Jon Nicholson of the Sanger Special Projects Group, who helped with aspects of the networking infrastructure and SAN design.

The publication costs of this article were defrayed in part by payment of page charges. This article must therefore be hereby marked "advertisement" in accordance with 18 USC section 1734 solely to indicate this fact.

REFERENCES

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. 1990. Basic local alignment search tool. *J. Mol. Biol.* **215:** 403–410.

Fox, G., Williams, R., and Messina, P. 1994. *Parallel computing works*. Morgan Kaufmann, San Francisco, CA.

Knaff, A. 2003. Udpcast. *http://udpcast.linux.lu*.

Moore, G. 1965. Cramming more components onto integrated circuits. *Electronics* **38:** 114–117.

Needleman, S.B. and Wunsch, C.D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* **48:** 443–453.

Pearson, W.R. and Lipman, D.J. 1988. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci.* **85:** 2444–2448.

Sanger, F. and Coulson, A.R. 1975. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. *J. Mol. Biol.* **94:** 441–448.

Smith, T.F. and Waterman, M.S. 1981. Identification of common molecular subsequences. *J. Mol. Biol.* **147:** 195–197.

Received August 8, 2003; accepted in revised form March 11, 2004.