Introduction

MySQL Cluster is an enterprise-grade, scalable, and highly available clustering product from MySQL AB, released under the MySQL dual license. MySQL Cluster is a share-nothing cluster with no single point of failure, and it is capable of running on inexpensive commodity hardware. MySQL Cluster allows the construction of cheap, scalable, and exceptionally reliable database clusters, without additional expensive specialized hardware or software. This makes MySQL Cluster a unique product.

It is possible to achieve telco-grade "five nines" uptime (99.999%) by using MySQL Cluster if you plan, deploy, and manage your cluster correctly. Managing a cluster in MySQL Cluster is a big step up from managing a classic MySQL server, and this book explains how MySQL Cluster works, taking you through the process of installing a cluster and explaining how best to manage it.

This book is *not* a "quick-start" guide. Simple guides on how to set up a very basic cluster can be found at MySQL's excellent developer site, http://dev.mysql.com. This book explains everything you need to know as a new cluster administrator. It assumes that you have basic knowledge of how MySQL and Linux in general work. Although some other platforms are supported (note that Windows is not included on this list), we do not explicitly cover these; the process for these platforms is very similar. We strongly recommend that you deploy your cluster using a recent version of Linux.

This book requires a little more patience and time than most introductory books simply due to the complexity of the underlying subject matter. You will experience a massive learning curve as you learn more and more about MySQL Cluster, and this book makes that curve significantly steeper. Experience shows that users who actually understand what they are doing rather than just following sample commands make far better administrators, and this is especially true with a complex piece of software such as MySQL Cluster.

This introduction first introduces you to MySQL Cluster and gives you a rough idea of its capabilities and limitations. It then introduces what this book covers.

Introduction to Database Clustering

In a classic MySQL database, rows of data are arranged in columns to form tables. These tables are stored in files on the disk of the database server, and queries are made to them. If the server crashes, the database goes down. If the load from all the queries gets too large, the only solution is to make the server more powerful.

Clustering spreads processing over multiple servers, resulting in a single redundant (that is, not reliant on any one single machine) and scalable (that is, you can add more machines) solution. A cluster in MySQL Cluster consists of a set of computers that run MySQL servers to receive and respond to queries, storage nodes to store the data held in the cluster and to process the queries, and one or more management nodes to act as a central point to manage the entire cluster. There are many reasons for clustering a database and several different methods of clustering.

The distinction between scaling up and scaling out is worth explaining. *Scaling up* is the traditional method that is used when the current hardware hits its limit: Administrators simply upgrade the hardware within the current server or replace it with a more powerful unit. *Scaling out* means that you *add* more servers while keeping your old hardware still in place. Scaling out is generally considered superior in terms of cost and reliability. However, it is much more difficult to set up, and software costs can be higher. Clusters in MySQL Cluster scale out (that is, they take what normally is powered by one server and spread it out over multiple servers). They are far more complicated to run than a standard MySQL server, but once correctly deployed, they provide the most cost-effective and reliable database cluster available. It is always worth remembering that support contracts are available from MySQL AB for deployments that require the additional guarantee of 24x7 support.

Why Cluster a Database?

There are two main reasons to cluster a database. The first is to allow the complete failure of individual servers within the cluster, without any downtime to the users of the database (that is, redundancy). The second is to allow more hardware to be added to the cluster—transparently to the users of the database—to increase performance (that is, scalability). This allows the database to be run on a large number of inexpensive machines, and it allows inexpensive machines to be added to the cluster as and when they are required.

There are other advantages of a cluster in MySQL Cluster compared to alternatives such as separate database servers with partitioned data, advanced replication setups, and other database clusters, including easier management (for example, all nodes in a cluster can be controlled from one single machine), lower costs, and greater reliability. It is also much easier to back up a large cluster than lots of separate database servers each serving up a partition of the data.

Types of Clustering

There are two main methods of clustering a database: shared-nothing and shared-disk clustering. There are also shared-memory clusters, which you can use if you use Dolphin SCI Interconnects with MySQL Cluster, which is covered later on in this book.

Shared-Disk Clustering

Shared-disk clustering means that multiple database servers actually read from the same disk (for example, a RAID array within a storage area network [SAN]). The challenge with this model is ensuring that no node undoes the work of another, which typically requires global memory and lock management. The lock manager must communicate with all the nodes on every transaction. As you might expect, the lock manager can quickly become the limiting factor in shared-disk scalability! Shared-disk clustering is used by Oracle Parallel Server and can be used by IBM's DB2.

Shared-Nothing Clustering

Shared-nothing clustering was developed in response to the limitations of shared-disk clustering. Each server owns its own disk resources (that is, they share nothing at any point in time). To maintain high availability, the cluster software monitors the health of each server in the cluster. In the event of a failure, the other servers in the cluster take over the job from the crashed server, transparently to the user of the database. This provides the same high level of availability

as shared-disk clustering, and it provides potentially higher scalability because it does not have the inherent bottleneck of a lock manager.

Shared-nothing clustering is generally considered superior to shared-disk clustering; however, it clearly requires far more complex software. MySQL clustering is shared-nothing clustering.

Replication Versus Clustering

MySQL has a feature known as *replication* that many confuse with clustering. *Replication* is server-implemented technology that protects against database failure by mirroring changes on a secondary server. It also provides a second access point for data that can protect against link failures and can share the load in high-traffic situations. However, the slave(s) must be read-only, so this still leaves the problem of a single point of failure as well as requiring a very powerful master machine because this machine must deal with all the write queries as well as the queries from the multiple slave machines in order to keep them all in sync. It is therefore not particularly scalable (the number of write operations that can be conducted is determined by the hardware of the master node) or highly available (if the master node dies, in most situations, at the very least, all write operations will fail).

There are many problems with trying to make replication highly available—mainly problems with primary keys. Replication is also asynchronous, which means that data is transmitted intermittently rather than in a steady stream. While there are "botches" to get replication to act as a cluster, MySQL Cluster is the only true solution for situations that require high availability and scalability.

Hardware Considerations with MySQL Cluster

MySQL Cluster is an *in-memory system*, which means it operates out of system RAM. For reasons that are explained later, the following is a rough guideline of the RAM required for a simple cluster:

```
Maximum Potential Size of One Row x1.1 x 2 x number of rows
```

MySQL Cluster in MySQL 4.1 and 5.0 is not efficient at storing `VARCHAR`, `TEXT`, or `BLOB` fields (it treats them all as full-length fields), although this problem is due to be fixed in version 5.1. You also need to include RAM for indexes, and you will see how to calculate the storage requirements in more detail later on. Suffice it to say for now that MySQL Cluster is a RAM-heavy application, and if you have a large database, you need a very large amount of RAM or a very large number of nodes: you can, of course, spread the large total amount of RAM required over many different physical servers.

Despite the fact that a cluster is RAM intensive, there is still a lot of disk activity because each member of the cluster keeps a fairly recent copy of its data on a local disk so that if all nodes crash (complete cluster shutdown) the cluster can recover fairly quickly. Therefore, for really optimum performance, RAID 1 and/or SCSI drives are recommended.

MySQL AB recommends the following hardware for servers acting as storage nodes in the cluster:

- **Operating system—** Linux (Red Hat, SUSE), Solaris, AIX, HP-UX, Mac OS X
- **CPU—** 2x Intel Xeon, Intel Itanium, AMD Opteron, Sun SPARC, IBM PowerPC
- **Memory—** 16GB RAM
- **Hard disk drive—** 4x 36GB SCSI (RAID 1 controller)
- **Network—** 1-8 nodes for Gigabit Ethernet; 8 or more nodes for Dedicated Cluster Interconnect

Networking Considerations with MySQL Cluster

A significant amount of traffic is transferred between servers within a cluster, so any latency between nodes is a very bad thing for performance. For this reason, MySQL Cluster is not suited to geographically diverse clusters. Rather, MySQL Cluster is intended to be used in a high-bandwidth environment, with computers typically connected via TCP/IP. Its performance depends directly on the connection speed between the cluster's computers. The minimum connectivity requirements for clusters include a typical 100Mbps Ethernet network or the equivalent. Gigabit Ethernet should be used if available.

Ideally, all the nodes in the cluster should be connected to a switch that is devoted just to the cluster traffic. (There are other reasons to take this approach, apart from performance, as you will see later on.) For enhanced reliability, you can use dual switches and dual cards to remove the network as a single point of failure; many device drivers support failover for such communication links.

MySQL Cluster supports high-speed Scalable Coherent Interface (SCI) interconnects. For clusters with eight or more servers, SCI interconnects are recommended. We will talk about these in more detail later on.

Why Use a MySQL Cluster?

Using MySQL Cluster has many advantages over running standalone MySQL servers. The following sections discuss three advantages—high availability, scalability, and performance—and briefly touch on some other advantages.

High Availability

Many administrators of MySQL servers are looking for enormous uptimes. The maximum that it is sensible to aim for in most cases is five nines (that is, 5.3 minutes downtime a year). This simply is not possible without redundancy, and, as we have seen, the only practical method of redundancy for MySQL is to use a cluster. This is because any one machine—regardless of how many "redundant" power supplies, hard drives, and CPUs it has—will fail occasionally; there is no such thing as an invincible server! The only way to eliminate this problem is to use share-nothing clustering. This way, it takes more than one failure to bring down the cluster, and the probability of an unusual event (such as a power supply or disk drive failure) occurring on two machines at exactly the same time is small enough to ignore. (Of course, you can design a cluster that can survive three or more concurrent failures, but you take a performance hit for this sort of setup.)

High availability means being able to suffer power supply or hard drive failures without any downtime to your users. It means being able to take servers down for upgrades while your database remains up. It means upgrading the database software without any downtime. All this is possible only with a clustering product such as MySQL Cluster. Note that some of these features—such as upgrades without downtime—are currently not supported within MySQL Cluster but are high up on the to-do list.

Scalability

Scalability is important because it allows you to increase your hardware costs as your database is used more. Put simply, it means that when you launch your database, you can start off with not very much hardware, and the hardware you use can be inexpensive. If you find that demand for your data is enormous, you can add a lot more hardware very quickly. At the moment, the addition of extra nodes requires a total cluster restart, but this should be fixed in MySQL 5.1.

One of MySQL Cluster's great features is near-linear scalability, so a two-server cluster will handle almost exactly half the load of a four-server cluster.

Performance

Sometimes MySQL Cluster does not result in higher performance, but in many cases it does. Queries that can be split up and run in parallel run much faster on a cluster in MySQL Cluster than on a single MySQL box because MySQL Cluster

runs the bits in parallel on different storage nodes. If you have queries that can not be split up and run in parallel, MySQL Cluster has to run them on one storage node, and clearly this is slower due to the additional networking overheads of a cluster.

Other Advanced Features

One problem with MySQL is the difficulty of making backups without locking files for fairly significant periods of time. To combat this problem, MySQL Cluster features hot online backups, which allows administrators to make backups while a cluster is running, without locking tables. This is one of the many features that make MySQL Cluster a product that can compete with vastly more expensive databases, such as Oracle and IBM DB/2.

MySQL Cluster features extremely fast (subsecond) failover in the event of a node failure. It features good management tools that, with the help of this book, allow administrators to manage a cluster with ease.

MySQL Cluster also supports simple parallel query processing. The cluster engine used for MySQL Cluster also provides full support for transactions; indeed, any nontransaction queries automatically have transactions wrapped around them.

Cluster Terminology

There are several significant terms that any administrator must come to grips with early on when using MySQL Cluster. The three different types of nodes are storage, SQL, and management nodes. The following sections discuss their different roles within a cluster. We will also talk about the differences between the storage engine within a cluster compared to two other common engines: MyISAM and InnoDB.

How a MySQL Cluster Works

MySQL Cluster consists of standard MySQL daemons and special daemons that control the storage and execution of queries in the background. These two different parts can run on the same servers or different servers. However, there must be at least one standard MySQL server and two background nodes running, spread out over at least two servers. A third server is always required as an arbitrator (as you shall see shortly), so the minimum number of servers on which you can set up a highly available cluster in MySQL Cluster is three.

Note

Many people find the requirement for three servers impossible to believe and try to set up two-server clusters. However, you must have at least three physical servers to set up a highly available cluster in MySQL Cluster. It is not possible to do with two servers. We explain the reason for this later on in the book.

In simple terms, a query comes into a standard MySQL daemon and is then sent to a background daemon (that is, a "storage node"). This daemons decides whether to answer the query alone or with other storage nodes (for greater speed) if the query is read-only, or if the query is a write or update query, it initiates the write as part of a transaction on all storage nodes and undergoes a two-phase commit process to ensure that all the other storage nodes have received and acted on the write.

Different Parts of a Cluster in MySQL Cluster

MySQL Cluster has three separate types of nodes (that is, services that form part of a cluster): storage nodes, SQL nodes, and management nodes. Nodes run on servers, and you can run multiple nodes on one physical server, subject to the limitations discussed later in this introduction.

Storage Nodes

Storage nodes store the fragments of data that make up the tables held in a cluster and do the early work in processing queries. Storage nodes require a large amount of RAM and relatively high-performance machines. The bulk of the processing is done on the storage nodes, and these are the nodes that an administrator spends most of his or her time tweaking. They are completely controlled from the management node via the configuration file and management client.

SQL Nodes

SQL nodes, which run on standard MySQL servers, are the nodes that applications can connect to. In small, simple clusters, these nodes are often run on the same physical servers as the storage nodes. Essentially, SQL nodes provide the "face" to the cluster and operate exactly as standard MySQL servers. They then connect to the storage nodes behind, which do the early processing on the queries and return the results for final processing at the SQL nodes. The amount of processing done on the SQL and storage nodes changes from query to query.

Management Nodes

Management nodes have two important roles: First, in most setups, they act as arbitrators if there are any networking problems between cluster nodes to decide which part of the cluster should remain alive. Second, they also are required when starting any other node in a cluster (they hold the configuration that each cluster node requires each time it starts), and they manage online backups. You can stop and restart storage and management nodes by issuing a signal from the management console, and you use the management server to get information about the current status of the cluster. You also use the management client as a central log of what is going on with the cluster. A cluster typically has one management node, although it is possible to have multiple management nodes.

The MySQL Cluster Storage Engine

You may be familiar with the classic storage engines of MySQL: MyISAM and InnoDB. MyISAM is very good at read-heavy environments, and InnoDB has transaction and other advanced feature support. MySQL Cluster uses neither of these; instead, it uses a transaction engine called Network Database (NDB). NDB was designed specifically for MySQL Cluster and for distribution over multiple storage nodes. It is in-memory (RAM) and so is very fast because it does not have the traditional bottlenecks of disk I/O.

MyISAM Versus NDB

Table IN.1 shows the key differences between MyISAM, the default storage engine in MySQL, and NDB, the only available storage engine with MySQL Cluster.

Table IN.1. MyISAM Versus NDB

Feature	MyISAM	NDB
Supports multistatement transactions and rollbacks	No	Yes
Supports full-text indexes	Yes	No
Can use hash lookups	No	Yes
Supports Unicode from version	4.1	5.0
Can compress read-only storage	Yes	No
Supports foreign keys	Yes	No
Uses a lot of RAM and has a lot of network traffic	No	Yes

InnoDB Versus NDB

Table IN.2 shows the key differences between InnoDB, the storage engine used by those requiring transactions within MySQL, and NDB, the only available storage engine with MySQL Cluster. InnoDB is in many ways more similar to NDB than MyISAM.

Table IN.2. InnoDB Versus NDB

Feature	InnoDB	NDB
Supports foreign key constraints	Yes	No
Supports Unicode from version	4.1.2	5.0
Uses a lot of RAM and has a lot of network traffic	No	Yes

Limitations of NDB

NDB has many limitations that are very easy to forget. Some databases cannot convert to NDB without significant modification, and often while importing a large existing database, you meet one of these limitations. Typically, as long as you can work out what limitation you have hit, there are ways around whatever problem you have met, but you should be aware that this is not always the case. The following are some of the possibilities:

- Database names, table names, and attribute names cannot be as long in NDB tables as with other table handlers. In NDB, attribute names are truncated to 31 characters, and if they are not unique after truncation, errors occur. Database names and table names can total a maximum of 122 characters
- NDB does not support prefix indexes; only entire fields can be indexed.
- A big limitation is that in MySQL 4.1 and 5.0, all cluster table rows are of fixed length. This means, for example, that if a table has one or more `VARCHAR` fields containing only relatively small values, more memory and disk space will be required when using the NDB storage engine than would be for the same table and data using the MyISAM engine. This issue is on the "to-fix" list for MySQL Cluster 5.1.
- In NDB, the maximum number of metadata objects is limited to 20,000, including database tables, system tables, indexes, and BLOBs (binary large objects). This is a hard-coded limit that you cannot override with a configuration option.
- The maximum permitted size of any one row in NDB is 8KB, not including data stored in BLOB columns (which are actually stored in a separate table internally).
- The maximum number of attributes per key in NDB is 32.
- Autodiscovery of databases is not supported in NDB for multiple MySQL servers accessing the same cluster in MySQL Cluster. (You have to add each database manually on each SQL node.)
- MySQL replication does not work correctly in NDB if updates are done on multiple MySQL servers; replication between clusters is on the feature list for MySQL 5.1.
- `ALTER TABLE` is not fully locking in NDB when you're running multiple MySQL servers.
- All storage and management nodes within a cluster in NDB must have the same architecture. This restriction does not apply to machines simply running SQL nodes or any other clients that may be accessing the cluster.
- It is not possible to make online schema changes in NDB, such as those accomplished using `ALTER TABLE` or `CREATE INDEX`. (However, you can import or create a table that uses a different storage engine and then convert it to NDB by using `ALTER TABLE tbl_name ENGINE=NDBCLUSTER;`.) `ALTER TABLE` works on occasions, but all it does is create a new table with the new structure and then import the data. This generally causes an error as NDB hits a limit somewhere. It is strongly recommended that you not use `ALTER TABLE` to make online schema changes.
- Adding or removing nodes online is not possible in NDB. (The cluster must be restarted in such cases.)
- The maximum number of storage nodes within an NDB cluster is 48.
- The total maximum number of nodes in a cluster in MySQL Cluster is 63. This number includes all MySQL servers (that is, SQL nodes), storage nodes, and management servers.

Data Partitioning

Data partitioning is an important concept in MySQL Cluster because it explains how a cluster splits up the data that is fed into it among the various storage nodes to achieve high performance and redundancy. Full partitioning, as described here, is available only in MySQL Cluster 5.1. MySQL Cluster 5.0 data is partitioned between storage node groups in a fairly random way.

A partitioned table is an abstract table that implements a table by making use of one stored table for each partition in the table (that is, it splits up one large table into lots of smaller tables). A partition function is a nonconstant and nonrandom function (typically a hash function) of one or more fields in the table; in MySQL Cluster, it is always the primary key. It cannot contain a query but can contain any scalar expression. Currently, the function needs to return an integer result. In addition, the partition function should be relatively simple because it will be evaluated very often in queries.

It is worth bearing in mind that MySQL Cluster automatically adds a primary key to any table that does not have one because it requires one for partitioning.

Benefits of Partitioning

Partitioning is essential in all clusters except clusters where the number of replicas is equal to the number of storage nodes because somehow the cluster must split the data equally across all the node groups. For example, if there are four nodes and two copies of every piece of data (that is, replicas), each node holds half the total data, and a partitioning function is required to decide where each piece of data can be found.

Certain queries can therefore be much more efficient. For example, consider the following query:

```
SELECT field1 FROM table1 WHERE id = 46;
```

If `id` were the primary key, the cluster would know immediately after running the partitioning function on the integer 46 which node group to send this query to for fast processing because the other node groups could never contain any records that would satisfy the `WHERE` clause.

Partitioning in general lays a foundation for parallel query support (that is queries executed on multiple nodes at the same time). For example, it is very simple to parallelize the query:

```
SELECT SUM(some_data) WHERE some_data > 100;
```

This query can be executed in parallel on each node, and the total sum is the sum of the query executed against each individual partition, resulting in the same answer being returned to the application much more quickly.

Synchronous Replication

Synchronous replication means that queries do not complete until the changes have been applied to all the servers involved. This has the benefit of guaranteeing that all servers have consistent copies of the data, but it can be a real performance problem, even though it eliminates the time-consuming operation of re-creating and replaying log files required by shared-disk architectures to fail over successfully.

This effectively means that when a MySQL Cluster node receives a write query (that is, a query that requires it to change its fragment of data), it issues the query on both itself and the other node(s) that holds this fragment of data. A transaction coordinator, chosen randomly from all the storage nodes, waits for the other node(s) to confirm acceptance of the transaction before issuing a commit and also confirming to the SQL node that issued the query that the transaction completed successfully.

Node Groups

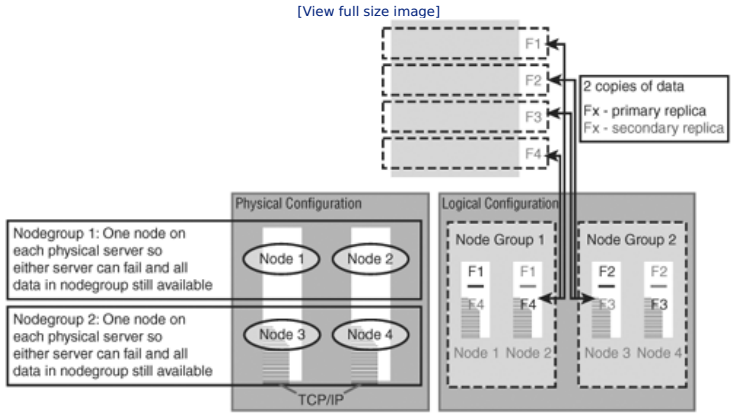
MySQL Cluster automatically splits nodes into node groups. The size of each node group is determined by the number of copies of each fragment of data that the cluster holds (which is configurable but is typically set to 2). So, if you have two copies of data, each node group has two nodes in it. As long as one node within each node group survives, the cluster will survive. Each node within a node group has identical data; consequently, as long as one node in each node group is alive, the cluster remains up because it has a complete set of data.

A Practical Example of Partitioning and Node Groups

Figure IN.1 shows two servers running four storage nodes on two physical machines. Physical Machine 1 is running Storage Node 1 and Storage Node 3, and Physical Machine 2 is running Storage Node 2 and Storage Node 4.

Figure IN.1. Two physical servers, with two storage nodes on each server, for a total of four storage nodes. With two replicas of each piece of data, we have two node groups. Either server can fail, and the database will remain up.

[View full size image]



The four storage nodes are split into two node groups, with Storage Nodes 1 and 2 making up Node Group 1 and Storage Nodes 3 and 4 making up Node Group 2.

Already, you can see that each physical server has one node from each node group on it—so a failure of either server still leaves one complete node group working on the other server.

Four items of data, F1 through to F4 (Fragments 1 through 4) are stored on this simple cluster. Each fragment of data has two copies—within the same node group. So Data Fragment 1 has a copy on Storage Node 1 (which is on Physical Server 1) and also a copy on Storage Node 2 (which is on Physical Server 2). You should notice how this is redundant: Either physical server could die, and the piece of data would remain available. You can put your finger over any one node in Figure IN.1 to convince yourself.

Network Partitioning and Arbitration

There is a fundamental problem with all clusters: If they get split in half, one half must shut down (or at least enter read-only mode) to prevent the two remaining halves from continuing to work and getting separate copies of the database, which would be difficult or impossible to merge at a later date. This problem is known as the "split-brain" problem.

MySQL solves this problem in a very simple way. First, as mentioned earlier, you must always have at least three physical servers in a cluster in MySQL Cluster for your cluster to be highly available; it is not possible to set up a highly available cluster with two (because otherwise, the cluster would be subject to the split-brain problem). A simplified explanation of how MySQL solves this problem goes like this: When nodes go down in a cluster, you have either majority rules (that is, if more than half of the nodes can see each other, they are the cluster) or, in a case in which you have an even number of nodes (such as 4) and only two nodes can see each other, it's whoever has the arbitrator. Note that the arbitrator is typically the management node. Note that we refer to a highly available cluster here: It is technically possible to set up a cluster with two or even one node, but doing so is pointless because it means you miss out on the main benefit of MySQL Cluster—high availability—and yet suffer all the overheads of a cluster.

A more complicated explanation of split-brain scenarios is required when you consider that the cluster is split into node groups. When all nodes in at least one node group are alive, it is not possible to get network partitioning. In this case, no other cluster half can form a functional cluster (because it would be missing the set of data that is in the node group that can still see the other). The problem comes when no node group has all its nodes alive because in that case, network partitioning is technically possible. In this situation, an arbitrator is contacted. The nodes in the cluster select the arbitrator before the node failure (during the startup process) so that all nodes will contact the same arbitrator. Normally the arbitrator would be the management server, but it can also be configured to be any of the MySQL servers in the cluster. The arbitrator will accept only the first cluster half to contact to survive. The second half will be told to die. Nodes that cannot connect to the arbitrator after a certain period of time automatically die, thus preventing a split-brain scenario.

The problem with network partitioning is that a highly available cluster can be formed only if at least three computers are involved in the cluster. One of those computers is needed only to handle the arbitrator function, so there is no need for high performance of any kind in this computer. This is because if you have only two nodes and they are unable to contact each other, which half should die? How does one node know if the other node has crashed or suffered a hardware failure or if there is a networking problem between them? It can't, so in such a cluster, both nodes die.

A Brief History of MySQL Cluster

The initial design, prototypes, and research for MySQL Cluster were completed between 1991 and 1996 by a company

called Alzato that was started by Ericcson. Most of the code today originates from 1996. The first demo of MySQL Cluster came in 1997, and since then, the feature set and reliability have increased massively. Version 1.0 came out in April 2001, with Perl DBI support. In version 1.4, in 2002, work on node recovery was completed. 2003 brought ODBC, online backup, unique indexes, and more. The most recent features include online upgrades and ordered indexes.

MySQL Cluster is now included with the MySQL-Max package and is also available as a separate package to download from www.mysql.com.

What's New in MySQL Cluster 5.0

All the following features are included in MySQL Cluster version 5.0:

- **Push-down conditions—** A query such as the following:

  ```
  SELECT * FROM t1 WHERE non_indexed_attribute = 1;
  ```

  uses a full table scan, which under older versions of MySQL Cluster would result in the entire `non_indexed_attribute` field in the t1 table being sent over the network to the SQL node and then each row being compared with 1. In Cluster 5.0 and later, the condition will be evaluated in the cluster's data nodes. Thus it is not necessary to send the records across the network for evaluation, which clearly increases performance significantly. This feature is disabled by default and can be enabled with the command `SET engine-condition-pushdown=On`.

- **Decreased `IndexMemory` usage (compared to earlier versions)—** In MySQL 5.0, each record consumes approximately 25 bytes of index memory, and every unique index uses 25 bytes per record of index memory because there is no storage of the primary key in the index memory. You should be aware that unique indexes also consume `DataMemory`, as explained in more detail later on.

- **New caches and optimizations—** The query cache has now been enabled, and many new optimizations have been introduced since version 4.1.

- **An increase in some hard limits—** Some hard limits have been increased. For example, there has been a significant increase in the limit on number of metadata objects from 1,600 to more than 20,000.

What's New in MySQL Cluster 5.1

All the following features are planned for release in MySQL Cluster 5.1:

- **Parallel processing for greater performance—** The amount of parallel processing that takes place on each node will be increased, and users will have the ability to define their own partitions and subpartitions.

- **Correct handling of variable-width fields—** In version 5, a column defined as `VARCHAR(255)` uses 260 bytes of storage, independent of what is stored in any particular record. In MySQL 5.1 Cluster tables, only the portion of the field actually taken up by the record will be stored. In many cases, this will make possible a reduction in space requirements for such columns by a factor of five.

- **Replication between clusters—** MySQL also hopes to integrate replication into clustering so you will be able to have replication slaves within a cluster.

- **Disk-based tables—** Disk-based tables are planned and are in early testing at the time of this writing; they will remove the current limitation that you must have at least 2.1 times your database size in RAM to run MySQL Cluster. However, you will still need indexes in RAM, and disk-based tables will likely reduce performance significantly.

Designing a Cluster

The following sections cover the essential things you should consider when designing a cluster. It is very important to design a cluster correctly before you start trying to use it; trying to run a cluster with two machines in different offices connected via the Internet will not work, nor will trying to import 2GB of data if you have only 1GB of RAM.

Networking

You need to connect all your nodes with at least a 100Mbps network connection. You are strongly advised to use a dedicated network for cluster traffic (that is, have two network cards in each node, eth0 and eth1, and use one of them for MySQL Cluster and the other for everything else). If you plan on having a cluster with a significant amount of traffic, you should use Gigabit Ethernet.

You should not try to set up cluster nodes all over a WAN such as the Internet. It won't work—nodes constantly communicate with each other using small pieces of information called heartbeats. The main role of a heartbeat is to ask the question "are you alive?" The receiving node simply sends another one back. As long as each node gets a heartbeat from each other node within the timeout value specified in the main configuration file, the cluster remains up. In a WAN, a large number of these heartbeats go missing or get delayed during transit, and the cluster ends up breaking very quickly.

Determining the Number of Replicas

A major decision is how redundant and fast you want your cluster; the more redundant and fast it is, the more RAM and hardware you need. The parameter `NumberOfReplicas` determines how many times MySQL Cluster stores each piece of data. To have a highly available cluster, you need at least two replicas, and this is the conventional value.

Just having more replicas does not automatically mean that you achieve higher performance. In fact, the law of diminishing returns kicks in fairly quickly, particularly if you have a write-heavy system (because each write must be committed `NumberOfReplicas` times before the transaction can be closed to the client, and although this happens in parallel, the more transactions that are going on, the longer it is likely to take), and it is unusual to have a value greater than `4` for `NumberOfReplicas`.

Determining the Amount of RAM

Having decided how many replicas you want, you need to decide how many nodes you need. The most important factor in this is RAM; you need to find your total RAM requirement and then divide that by the amount of RAM you have in each machine to work out how many nodes you require. As you have already seen, this is a rough estimate of RAM usage for a "typical" cluster:

```
Data Size x NumberOfReplicas x 1.1
```

It should be noted that your RAM usage can be massively larger in several situations, and if you have lots of indexes, you should consider changing 1.1 to something much larger, such as 1.5.

As a practical example, say you have a row size of 100KB and 100,000 rows. That is just under 1000MB or 1GB of data size. Say you have decided that you want four replicas. The rough estimate of RAM would be 1,000 x 4 x 1.1 =4,400, or 4.4GB.

If you have machines that can take 2GB of RAM, you want 2.2 machines. You clearly can't have 2.2 machines, so you round this up to 3. However, your total number of nodes should be a multiple of `NumberOfReplicas`; 3 is a suboptimal number of servers, so we would settle on 4 machines.

You might consider four servers slightly excessive, but there are several points to bear in mind:

- Although there may be 2GB of RAM per storage node, some of this is used by the operating system, and a certain amount of free RAM is important if you want a stable system.
- It is always good to have a significant amount of spare RAM within a cluster so you do not run the risk of running out of RAM and crashing if the amount of data increases for some reason.
- It is safest to round up at every stage. You will use more RAM than you anticipate.

It is strongly recommended for performance reasons that you have 2, 4, 8, or 12 nodes for performance reasons and not any number in between these. If you use other numbers, you will find that the partitioning function struggles to evenly partition the data between nodes, so you will not get well-balanced partitions.

Determining the Amount of Disk Space

You want plenty of disk space free on the storage nodes. The disk will be used for backups as well as regular commits of the data, which are used if the cluster suffers a fatal crash.

If you want fast backups or have a write-heavy application, you should certainly consider using SCSI drives or some sort of RAID solution. If you have only two replicas, you might want to ensure that both nodes remain up for the vast majority of the time for performance reasons, so RAID may be important here as well. Although I/O performance does not directly increase the performance of a cluster, it may help reduce the overall system load if other processes are not waiting around for disk access. If you can afford it, this is highly recommended.

Choosing an Operating System

MySQL Cluster has been run on many diverse operating systems, including Solaris and BSD, as well as Linux. If you want an easy life and just want a cluster to work, you should run all nodes on Linux. This will almost certainly get you the best possible performance and stability.

You can select any Linux distribution you like, although the operating system that most new administrators go with is something like Red Hat Enterprise 4, which is perfectly suited to this. CentOS (www.centos.org) provides a "binary compatible" (that is, identical) version of Red Hat Enterprise Linux (RHEL), which is also complexly free, so if you do not need the support offered by Red Hat, you can use CentOS. Other popular Linux distributions are SUSE and Debian. The chances are very high that whatever versions of Linux you choose, it will come with MySQL preinstalled.

The code for running MySQL Cluster on Windows is not yet complete, and you should not attempt to run MySQL Cluster in a production environment on Windows.

A Note on Hardware

You should ensure that all your storage nodes are as identical as possible. Although it is possible to have different amounts of RAM on each storage node, we do not advise it. If you have completely identical machines (apart from IP address), having the same amount of RAM on each machine will make management much simpler, and it will make rapid disaster recovery fairly easy. If you have differing amounts of RAM, you will only be able to use the smallest amount of RAM in any storage node on all storage nodes, so the extra RAM will probably just be wasted.

What This Book Covers

This book covers a lot of very complex topics. The following sections briefly explain what you can expect to find in each chapter. The book is designed so you can either dip in and out or read it cover to cover, depending on your existing knowledge, time available, and goals.

Chapter 1, "Installation"

Chapter 1 covers installing MySQL Cluster. This includes the basic installation process, either from binaries or from source. Chapter 1 covers the differences in the process between the various different versions, and it shows how to install the cutting-edge nightly builds. Although you'll learn about advanced configuration later, Chapter 1 introduces you to the various configuration files you will later become familiar with, when you use them to get your cluster up and running.

Chapter 1 also covers some common installation problems and the best methods for upgrading a cluster in a live environment. More complex installation scenarios, such as disaster recovery, are covered in later chapters but require a strong understanding of how to install the software.

Finally, Chapter 1 covers importing your current tables into the new cluster, as well as some of the common problems that can occur during this process. This process is more complex than it sounds, and you should not expect it to be as simple as you might have hoped, unless your table structure is exceptionally simple.

Chapter 2, "Configuration"

When you have a basic cluster installed, you need to be able to configure it. Chapter 2 introduces you to the hundreds of configuration options you can set to control all aspects of a cluster. It explains them all in clear English so you know exactly what you can configure and where.

Chapter 2 explains how to calculate the size of a database and how to work out how many storage nodes you require and how much RAM you need in each. It then explains the advanced configuration options you are likely to want to use at some stage. Finally, Chapter 2 explains all the complex options you are unlikely to need—just in case you are inquisitive, like to control everything, or come across a situation where you need to configure these options.

Chapter 2 also gives you some examples of configuration files that you can use as a base for your configuration, which should speed up the process of setting up a cluster.

Chapter 3, "Backup and Recovery"

Chapter 3 covers the topic of backing up and restoring a new cluster, including recovering from various scenarios. Many features have been added to MySQL Cluster that will make backing up and restoring a cluster a very neat and easy process.

MySQL Cluster includes support for hot online backups, and this is one of the major features that most MySQL administrators have been wishing for! Of course, no backup is good unless the disaster recovery process is smooth, and Chapter 3 covers this in detail, describing the different disasters you could potentially have to restore from and the best method to use in each case.

Chapter 4, "Security and Management"

Chapter 4 covers two important topics: advanced usage of the management console to help manage a cluster and securing a cluster. Securing a cluster is vital because MySQL Cluster provides absolutely no security, and you must add your own if you want to use it in a production environment.

Having a good working knowledge of all the commands you can issue to the management console is critical for efficient management of a cluster because it is the only sensible way of communicating with individual nodes.

Security is vital, and Chapter 4 discusses many different tricks and tips that are not actually part of MySQL Cluster. It is no exaggeration to say that there is not a single bit of security in MySQL Cluster, so you have to ensure that your cluster is secure by using other applications, such as firewalls, and additional hardware, such as dedicated networks for cluster traffic.

Chapter 5, "Performance"

After a cluster is installed, configured, backed up, and secured, it is then important to get as much performance out of it as you can. Chapter 5 shows you the many different ways of increasing the performance of a cluster.

There are so many different ways that you can optimize a database that Chapter 5 is vital reading. Because there are so many different uses for a database, some of the suggestions in Chapter 5 might not make sense in your situation, but you should be able to significantly increase the performance of your cluster if you follow a few of the suggestions. Performance is even more critical with a cluster than with a standard MySQL server. Whereas a standard MySQL installation will perform fairly well most of the time with a default installation, this is not the case with a cluster; if you do not make any effort to tweak the performance of a cluster, the chances are very high that you will get exceptionally poor performance or require expensive hardware to get good performance.

As well as showing you how to increase performance of a cluster, Chapter 5 also shows you how to stress-test the cluster with various different tools, and it also describes the different ways of automatically monitoring of a cluster so that you know when any dragons appear before they rear their heads and become apparent to your users. This is of critical importance if you plan to use a cluster in a production environment because there is no nice centralized GUI for MySQL Cluster; you have to write scripts and regularly check things manually to ensure that a cluster is working as you expect.

Chapter 6, "Troubleshooting"

Chapter 6 describes solutions to the most common problems users meet when using MySQL Cluster. You should check here before you post on the mailing lists or give up! Working with MySQL Cluster is frustrating at times, and Chapter 6 covers a few of the problems that many users run into time and time again.

Chances are that during your experimentation with MySQL Cluster, you will run into a problem that you may not understand. The trick is always to look for clues—error numbers in log files, for example—and then look for solutions either in Chapter 6 or on the Internet. Chapter 6 covers the basic troubleshooting steps you should carry out if you have a problem, including where error logs are likely to be held as well as how to ask for help or submit a bug report.

Chapter 7, "Common Setups"

Chapter 7 gives you some advice about some very common setups that many users have. Chances are that the scenario in which you plan to use your cluster is either identical or very similar to one of the setups in this chapter.

You may find it helpful, even if your eventual setup is not in this chapter, to try one of these setups first so you can get to know how it works before plunging in and designing your own setup.

Software Versions

Writing a book on any piece of software is a challenge, but on a brand-new piece of software such as MySQL Cluster, it is especially difficult due to the fantastic speed of change. New features are constantly added, and many current features are slightly changing all the time.

Most of the writing of this book was carried out before MySQL 5.0 left beta testing, but we expect that MySQL 5.0 will remain the stable version for some time to come. If you use version 5.1 when it becomes stable, we anticipate that it will be backward compatible, and where it is not, we try to mention it in this book so you should be able to follow this book with ease if you are using either MySQL Cluster 5.0 or 5.1.

More Sources of Information

Apart from this book, there are several other sources of information that we recommend if you want to explore something in more detail or if you get stuck:

- You can find the official MySQL Cluster documentation at http://dev.mysql.com/doc/mysql/en/ndbcluster.html.
- You can browse the mailing list archive at http://lists.mysql.com/cluster/.
- The MySQL forum contains a cluster forum that you can find at http://forums.mysql.com/list.php?25.

You will also find that there are a large number of posts about MySQL on the Internet, and with a quick Internet search, you can often quickly find people who have had the same problem you're facing.