Administrator's Guide

Abstract

This books explains how to create and manage VoltDB databases and the clusters that run them.

V4.0.2

Administrator's Guide

V4.0.2

This document was generated on January 26, 2014.

Table of Contents

List of Tables

Preface

This book explains how to manage VoltDB databases and the clusters that host them. It is intended for database administrators and operators, responsible for the ongoing management and maintenance of database infrastructure.

1. Structure of This Book

This book is divided into 6 chapters and 2 appendices:

- Chapter 1, *Managing VoltDB Databases*
- Chapter 2, *Preparing the Servers*
- Chapter 3, *Starting and Stopping the Database*
- Chapter 4, *Maintenance and Upgrades*
- Chapter 5, *Monitoring VoltDB Databases*
- Chapter 6, *What to Do When Problems Arise*
- Appendix A, *Server Configuration Options*
- Appendix B, *Snapshot Utilities*

2. Related Documents

This book does not describe how to design or develop VoltDB databases. For a complete description of the development process for VoltDB and all of its features, please see the accompanying manual *Using VoltDB*. For information on managing VoltDB databases using a graphical interface, see the *VoltDB Enterprise Manager Guide*. These and other books describing VoltDB are available on the web from http://www.voltdb.com/.

Chapter 1. Managing VoltDB Databases

VoltDB is a distributed, in-memory database designed from the ground up to maximize throughput performance on commodity servers. The VoltDB architecture provides many advantages over traditional database products while avoiding the pitfalls of NoSQL solutions:

- By partitioning the data and stored procedures, VoltDB can process multiple queries in parallel without sacrificing the consistency or durability of an ACID-compliant database.
- By managing all data in memory with a single thread for each partition, VoltDB avoids overhead such as record locking, latching, and device-contention inherent in traditional disk-based databases.
- VoltDB databases can scale up to meet new capacity or performance requirements simply by adding more nodes to the cluster.
- Partitioning is automated, based on the schema, so there is no need to manually shard or repartition the data when scaling up as with many NoSQL solutions.
- Finally, VoltDB Enterprise Edition provides features to ensure durability and high availability through command logging, locally replicating partitions (K-safety), and wide-area database replication.

Each of these features is described, in detail, in the *Using VoltDB* manual. This book explains how to use these and other features to manage and maintain a VoltDB database cluster from a database administrator's perspective.

1.1. Getting Started

The main difference between VoltDB and traditional database products from an administration perspective is that you must define the database schema *before* you start the database. There is no such thing as starting an "empty" database server.

To start a VoltDB database cluster, you need two files:

Application Catalog	The compiled database schema and stored procedures. The application catalog defines the logical structure of the database, including the tables, indexes, partitioning, and so on.
Deployment File	The configuration settings for the cluster. The deployment file defines the physical structure of the database cluster, including the size of the cluster, what availability, durability, and security options will be enabled, and so on.

In general, the application catalog is produced as part of the database development process, which is described in the *Using VoltDB* manual. This book assumes you have a completed catalog.

The deployment file, on the other hand, defines the run-time configuration of the cluster. Establishing the correct settings for the deployment file and physically managing the database cluster is the duty of the administrators who are responsible for maintaining database operations. This book is written for those individuals and covers the standard procedures associated with database administration.

1.2. Understanding the VoltDB Utilities

VoltDB provides several command line utilities, each with a different function. Familiarizing yourself with these utilities and their uses can make managing VoltDB databases easier. The three primary command line tools for creating, managing, and testing VoltDB databases are:

voltdb	Compiles application catalogs and starts the VoltDB database process. The voltdb command can also collect log files for analyzing possible system errors (see Section 6.2.1, “Collecting Log Files Using the Command Line” for details). The voltdb command runs locally and does not require a running database.
voltadmin	Issues administrative commands to a running VoltDB database. You can use voltadmin to save and restore snapshots, pause and resume admin mode, and to shutdown the database, among other tasks. The voltadmin command can be run remotely, performs cluster-wide operations and requires a running database to connect to.
sqlcmd	Lets you issue SQL queries and invoke stored procedures interactively. The sqlcmd command is handy for testing database access without having to write a client application. The sqlcmd command can be run remotely and requires a running database to connect to.

In addition to the preceding general-purpose tools, VoltDB provides several other tools for specific tasks:

csvloader	Loads records from text files into an existing VoltDB database. The command's primary use is for importing data into VoltDB from CSV and other text-based data files that were exported from other data utilities, The csvloader command can be run remotely and requires a running database to connect to.
dragent	Starts the database replication agent, initiating replication between a master and a replica database. The dragent command is run separately from the database servers but requires both a running master database and replica database.
snapshotconvert	Converts native snapshot files to csv or tabbed text files. The snapshotconvert command is useful when exporting a snapshot in native format to text files for import into another data utility. (This utility is provided for legacy purposes. It is now possible to write snapshots directly to CSV format without post-processing, which is the recommended approach.) The snapshotconvert command runs locally and does not require a running database.
snapshotverify	Verifies that a set of native snapshot files are complete and valid. The snapshotverify command runs locally and does not require a running database.

1.3. Management Tasks

Database administration responsibilities fall into five main categories, as described in Table 1.1, “Database Management Tasks”. The following chapters are organized by category and explain how to perform each task for a VoltDB database.

Table 1.1. Database Management Tasks

Preparing the Servers	Before starting the database, you must make sure that the server hardware and software is properly configured. This chapter provides a checklist of tasks to perform before starting VoltDB.

Basic Database Operations	The basic operations of starting and stopping the database. This chapter describes the procedures needed to handle these fundamental tasks.
Maintenance and Upgrades	Over time, both the cluster and the database may require maintenance — either planned or emergency. This chapter explains the procedures for performing hardware and software maintenance, as well as standard maintenance, such as backing up the database and upgrading the hardware, the software, and the database schema.
Performance Monitoring	Another important role for many database administrators is monitoring database performance. Monitoring is important for several reasons: • Performance Analysis • Load Balancing • Fault Detection This chapter describes the tools available for monitoring VoltDB databases.
Problem Reporting & Analysis	If an error does occur and part or all of the database cluster fails, it is not only important to get the database up and running again, but to diagnose the cause of the problem and take corrective actions. VoltDB produces a number of log files that can help with problem resolution. This chapter describes the different logs that are available and how to use them to diagnose database issues.

Chapter 2. Preparing the Servers

VoltDB is designed to run on commodity servers, greatly reducing the investment required to operate a high performance database. However, out of the box, these machines are not necessarily configured for optimal performance of a dedicated, clustered application like VoltDB. This is especially true when using cloud-based services. This chapter provides best practices for configuring servers to maximize the performance and stability of your VoltDB installation.

2.1. Server Checklist

The very first step in configuring the servers is making sure you have sufficient memory, computing power, and system resources such as disk space to handle the expected workload. The *VoltDB Planning Guide* provides detailed information on how to size your server requirements.

The next step is to configure the servers and assign appropriate resources for VoltDB tasks. Specific server features that must be configured for VoltDB to perform optimally are:

- Install required software
- Disable swapping
- Turn off TCP Segmentation
- Configure NTP (time services)
- Define network addresses for all nodes in the cluster
- Assign network ports

2.2. Install Required Software

To start, VoltDB requires a recent release of the Linux operating system. The supported operating systems for running production VoltDB databases are:

- CentOS V6.3 or later
- Red Hat V5.8 or later and V6.3 or later
- Ubuntu 10.4 and 12.4

It may be possible to run VoltDB on other versions of Linux. Also, an official release for Macintosh OS X 10.7 and later is provided for development purposes. However, the preceding operating system versions are the only fulled tested and supported base platforms for running VoltDB in production.

In addition to the base operating system, VoltDB requires the following software at a minimum:

- Java 7
- NTP
- Python 2.4 or later

Sun Java SDK 7 or later is recommended. OpenJDK 7 or later is also supported. Note that the VoltDB server requires Java 7; however, the Java client is also compatible with Java 6.

VoltDB requires the system clocks on all cluster nodes be synchronized to within 100 milliseconds. NTP, the Network Time Protocol, is recommended for achieving the necessary synchronization. NTP is installed and enabled by default on many operating systems. However, the configuration may need adjusting (see Section 2.5, “Configure NTP” for details) and in cloud instances where hosted servers are run in a virtual environment, NTP is not always installed or enabled by default. Therefore you need to do this manually.

Finally, VoltDB implements its command line interface through Python. Python 2.4 or later is required to use the VoltDB shell commands.

2.3. Disable Swapping

Swapping is an operating system feature that optimizes memory usage when running multiple processes. However, memory is a critical component of the VoltDB server process. Any contention for memory, including swapping, will have a very negative impact on performance and functionality.

To disable swapping on Linux systems, use the `swapoff` command. If swapping cannot be disabled for any reason, you can reduce the likelihood of VoltDB being swapped out by setting the kernel parameter `vm.swappiness` to zero.

2.4. Turn off TCP Segmentation

Under certain conditions, the use of TCP segmentation offload (TSO) and generic receive offload (GRO) can cause nodes to randomly drop out of a cluster. The symptoms of this problem are that nodes timeout — that is, the rest of the cluster thinks they have failed — although the node is still running and no other network issues (such as a network partition) are the cause.

Disabling TSO and GRO is recommended for any VoltDB clusters that experience such instability. The commands to disable offloading are the following, where N is replaced by the number of the ethernet card:

```
ethtool -K ethN tso off
ethtool -K ethN gro off
```

Note that these commands disable offloading temporarily. You must issue these commands every time the node reboots or, preferably, put them in a startup configuration file.

2.5. Configure NTP

To orchestrate activities between the cluster nodes, VoltDB relies on the system clocks being synchronized. Many functions within VoltDB — such as cluster start up, nodes rejoining, and schema updates among others — are sensitive to variations in the time values between nodes in the cluster. Therefore, it is important to keep the clocks synchronized within the cluster. Specifically:

- The server clocks within the cluster must be synchronized to within 100 milliseconds of each other when the cluster starts. (Ideally, skew between nodes should be kept under 10 milliseconds.)
- Time must not move backwards

The easiest way to achieve these goals is to install and configure the NTP (Network Time Protocol) service to use a common time host server for synchronizing the servers. NTP is often installed by default but may require additional configuration to achieve acceptable synchronization. Specifically, listing only one time server (and the same one for all nodes in the cluster) ensures minimal skew between servers. You can even establish your own time server to facilitate this. All nodes in the cluster should also list each other as peers. For example, the following NTP configuration file uses a local time server (myntpsvr) and establishes all nodes in the cluster as peers:

```
server myntpsvr burst iburst minpoll 4 maxpoll 4

peer voltsvr1 burst iburst minpoll 4 maxpoll 4
peer voltsvr2 burst iburst minpoll 4 maxpoll 4
peer voltsvr3 burst iburst minpoll 4 maxpoll 4

server 127.127.0.1
```

See the chapter on Configuring NTP in the *Performance Guide* for more details on setting up NTP.

2.6. Configure the Network

It is also important to ensure that the network is configured correctly so all of the nodes in the VoltDB cluster recognize each other. If the DNS server does not contain entries for all of the servers in the cluster, an alternative is to add entries in the /etc/hosts file locally for each server in the cluster. For example:

```
12.24.48.101   voltsvr1
12.24.48.102   voltsvr2
12.24.48.103   voltsvr3
12.24.48.104   voltsvr4
12.24.48.105   voltsvr5
```

2.7. Assign Network Ports

VoltDB uses a number of network ports for functions such as internal communications, client connections, rejoin, database replication, and so on. For these features to perform properly, the ports must be open and available. Review the following list of ports to ensure they are open and available (that is, not currently in use).

Function	**Default Port Number**
Client Port	21212
Admin Port	21211
Web Interface Port (httpd)	8080
Internal Server Port	3021
Log Port	4560
JMX Port	9090
Replication Port	5555, 5556, 5557
Zookeeper port	2181

Alternately, you can reassign the port numbers that VoltDB uses. See Section A.5, “Network Ports” for a description of the ports and how to reassign them.

Chapter 3. Starting and Stopping the Database

The fundamental operations for database administration are starting and stopping the database. The one significant difference between VoltDB and other database systems is that you do not start the database management system, or DBMS, separate from the database itself. When you start VoltDB, you start it with a specific database schema, as defined in the application catalog.

You also need to decide what database features you want to configure before starting the database. This includes the size of the cluster, what amount of replication you want to use to increase availability in case of server failure, and what level of durability is required for those cases where the database itself stops. These settings are defined in the deployment file.

This chapter explains how to configure the cluster physical structure and database features in the deployment file and how to start and stop the database.

3.1. Configuring the Cluster and Database

You specify the cluster configuration and what features to use in the deployment file, which is an XML file that you can create and edit manually. In the simplest case, the deployment file specifies how many servers the cluster has initially, how many partitions to create on each server, and what level of availability (K-safety) to use. For example:

```
<?xml version="1.0"?>
<deployment>
  <cluster hostcount="5"
           sitesperhost="4"
           kfactor="1"
  />
</deployment>
```

- The `hostcount` attribute specifies the number of servers the cluster will start with.
- The `sitesperhost` attribute specifies the number of partitions (or "sites") to create on each server. The optimal number of sites per host is affected by the number of processors per machine and is best determined by performance testing against the expected workload. See the chapter on "Benchmarking" in the *VoltDB Planning Guide* for details.
- The `kfactor` attribute specifies the K-safety value to use. The higher the K-safety value, the more node failures the cluster can withstand without affecting database availability. However, increasing the K-safety value increases the number of copies of each unique partition. High availability is a trade-off between replication to protect against node failure and the number of unique partitions, therefore throughput performance. See the chapter on availability in the *Using VoltDB* manual for more information on determining an optimal K-safety value.

In addition to the cluster configuration, you can use the deployment file to enable and configure specific database features such as export, command logging, and so on. The following table summarizes some of the key features that are settable in the deployment file.

Table 3.1. Configuring Database Features in the Deployment File

Feature	Example

Command Logging — Command logging provides durability by logging transactions to disk so they can be replayed during a recovery. You can configure the type of command logging (synchronous or asynchronous), the log file size, and the frequency of the logs (in terms of milliseconds or number of transactions).	<commandlog enabled="true"
	            synchronous="false">
	  <frequency time="300"
	             transactions="1000"/>
	</commandlog>

Snapshots — Automatic snapshot provide another form of durability by creating snapshots of the database contents, that can be restored later. You can configure the frequency of the snapshots, the unique file prefix, and how many snapshots are kept at any given time.	<snapshot enabled="true"
	          frequency="30m"
	          prefix="mydb"
	          retain="3" />

Export — Export allows you to write selected records from the database to an external target, which can be files, another database (through JDBC) or a custom export client. You can configure the type of export as well as other properties, which are specific to each target type. For example, the file target requires a specific type (or format) for the files and a unique identifier called a "nonce".	<export enabled="true" target="file"> <configuration> <property name="type">csv</property> <property name="nonce">mydb</property> </configuration> </export>

Security & Accounts — Security lets you protect your database against unwanted access by requiring all connections authenticate against known usernames and passwords. In the deployment file you can define the user accounts and paswords and what role or roles each user fulfills. Roles define what permissions the account has. Roles are defined in the application catalog.	<security enabled="true"/>
	<users>
	  <user name="admin"
	        password="superman"
	        roles="dev,ops"/>
	  <user name="mitty"
	        password="thurber"
	        roles="user"/>
	</users>

File Paths — Paths define where VoltDB writes any files or other disc-based content. You can configure the default root for all files as well as specific paths for each type of service, such as snapshots, command logs, export overflow, etc.	<paths> <voltdbroot path="/tmp/vroot" /> <snapshots path="/opt/archive" /> </paths>

3.2. Starting the Database

Once you have an application catalog and a deployment file, you start a VoltDB database cluster[1] for the first time using the **voltdb create** command. You issue this command, specifying the same application catalog, deployment file, and host on each node of the cluster. For example:

```
$ voltdb create myapplication.jar \           ❶
```

[1]When testing with a single server, several of the command line arguments have defaults and can be left out. However, in production when starting a multi-node cluster, the arguments are required.

```
--deployment=deployment.xml \ ❷
--host=voltsvr1 \             ❸
--license=~/license.xml       ❹
```

On the command line, you specify four arguments:

❶ The application catalog, which contains the schema for the database and any stored procedure definitions

❷ The deployment file, which specifies the physical layout of the cluster and configures specific VoltDB features

❸ One node of the cluster identified as the "host", to coordinate the initial startup of the cluster

❹ The license file (when using the VoltDB Enterprise Edition)

What happens when you start the database is that each server contacts the named "host" server. The host then:

1. Waits until the necessary number of servers (as specified in the deployment file) are connected
2. Creates the network mesh between the servers
3. Distributes the catalog and deployment file to ensure all nodes are using the same configuration

At this point, the cluster is fully initialized and the "host" ends its special role and becomes a peer to all the other nodes. All nodes in the cluster then write an informational message to the console verifying that the database is ready:

```
Server completed initialization.
```

3.3. Stopping the Database

How you choose to stop a VoltDB depends on what features you have enabled. For example, if you do not have any durability features enabled (such as auto snapshots or command logging), it is strongly recommended that you pause the database and take a manual snapshot before shutting down, so you preserve the data across sessions.

If you have command logging enabled, a manual snapshot is not necessary. However, it is still a good idea to pause the database before shutting down to ensure that all active client queries have a chance to complete and return their results (and no new queries start) before the shutdown occurs.

To pause and shutdown the cluster you can use the **voltadmin pause** and **shutdown** commands:

```
$ voltadmin pause
$ voltadmin shutdown
```

As with all **voltadmin** commands, you can use them remotely by specifying one of the cluster servers on the command line:

```
$ voltadmin pause --host=voltsvr2
$ voltadmin shutdown --host=voltsvr2
```

If security is enabled, you will also need to specify a username and password for a user with sysproc permissions:

```
$ voltadmin pause --host=voltsvr2 -u root -p Suda51
$ voltadmin shutdown --host=voltsvr2 -u root -p Suda51
```

Finally, if you are not using the durability features of automatic snapshots or command logging, you should perform a manual snapshot using the **save** command after pausing and before shutting down. Use the `--blocking` flag to ensure the snapshot completes before the shutdown occurs:

```
$ voltadmin pause
$ voltadmin save --blocking /tmp/voltdb backup
$ voltadmin shutdown
```

3.4. Restarting the Database

Restarting a VoltDB database is different than starting it for the first time. How you restart the database depends on what durability features were in effect previously.

If you just use the **voltdb create** command, you create a new, empty database. Because VoltDB keeps its data in memory, to return the database to its last known state — including its content — you need to restore the data from a saved copy.

If you are using automatic snapshots or command logging, VoltDB can automatically reinstate the data when you use the **voltdb recover** command:

```
$ voltdb recover --deployment=deployment.xml \
                --host=voltsvr1 \
                --license=~/license.xml
```

Because the schema is saved with the data, you do not need to specify an application catalog when recovering a previous database session. However, just as when starting a database for the first time, you must invoke the **recover** command on all nodes of the cluster before the database can start,

When you recover a VoltDB database, the cluster performs the same initial coordination activities as when creating a new database: the host node facilitates establishing a quorum and ensures all nodes connect. Then the database servers restore the most recent snapshot plus (if command logging is enabled) the last logged transactions. Once all data is restored, the database enables client access.

If you are not using automatic snapshots or command logging, you must restore the last snapshot manually. You do this with the following procedure:

1. Start a new database using the **voltdb create** command on each server as described in Section 3.2, “Starting the Database”.

2. Use the **voltadmin pause**, **restore**, and **resume** commands to pause the database, restore the last snapshot, and then resume client activity.

For example, if the last snapshot was saved in `/tmp/voltdb` using the unique ID `backup`, you can restore the data with the following commands:

```
$ voltadmin pause
$ voltadmin restore /tmp/voltdb backup
$ voltadmin resume
```

Chapter 4. Maintenance and Upgrades

Once the database is running, it is the administrator's role to keep it running. This chapter explains how to perform common maintenance and upgrade tasks, including:

- Database backups
- Schema and stored procedure updates
- Software and hardware upgrades

4.1. Backing Up the Database

It is a common safety precaution to backup all data associated with computer systems and store copies off-site in case of system failure or other unexpected events. Backups are usually done on a scheduled basis (every day, every week, or whatever period is deemed sufficient).

VoltDB provides several options for backing up the database contents. The easiest option is to save a native snapshot then backup the resulting snapshot files to removable media for archiving. The advantage of this approach is that native snapshots contain both a complete copy of the data and the schema. So in case of failure the snapshot can be restored to the current or another cluster using a single **voltadmin restore** command.

The key thing to remember when using native snapshots for backup is that each server saves its portion of the database locally. So you must fetch the snapshot files for all of the servers to ensure you have a complete set of files. The following example performs a manual snapshot on a five node cluster then uses scp to remotely copy the files from each server to a single location for archiving.

```
$ voltadmin save --blocking --host=voltsvr3 \
            /tmp/voltdb backup
$ scp 'voltsvr1:/tmp/voltdb/backup*' /tmp/archive/
$ scp 'voltsvr2:/tmp/voltdb/backup*' /tmp/archive/
$ scp 'voltsvr3:/tmp/voltdb/backup*' /tmp/archive/
$ scp 'voltsvr4:/tmp/voltdb/backup*' /tmp/archive/
$ scp 'voltsvr5:/tmp/voltdb/backup*' /tmp/archive/
```

Note that if you are using automated snapshots or command logging (which also creates snapshots), you can use the automated snapshots as the source of the backup. However, the automated snapshots use a programmatically generated file prefix, so your backup script will need some additional intelligence to identify the most recent snapshot and its prefix.

Finally, if you wish to backup the data in a non-proprietary format, you can use the **voltadmin save --format=csv** command to create a snapshot of the data as comma-separated value (CSV) formatted text files. The advantage is that the resulting files are usable by more systems than just VoltDB. The disadvantage is that the CSV files only contain the data, not the schema. These files cannot be read directly into VoltDB, like a native snapshot can. Instead, you will need to create a new database then use the csvloader utility to load individual files into each table to restore the database completely.

4.2. Updating the Database Schema

As an application evolves, the database schema often needs changing. This is particularly true during the early stages of development and testing but also happens periodically with established applications, as the database is tuned for performance or adjusted to meet new requirements. In the case of VoltDB, these updates may involve changes to the table definitions, to the indexes, or to the stored procedures. The following sections explain how to:

- Perform live schema updates
- Change unique indexes and partitioning using save and restore

4.2.1. Performing Live Schema Updates

There are two ways to update the database schema for a VoltDB database: live updates and save/restore updates. For most updates, you can update the schema while the database is running. To perform this type of live update, use the following steps:

1. Compile a new application catalog containing the updated schema and stored procedures
2. Use the **voltdb update** command, specifying the new catalog and existing deployment file, to update the database

For example:

```
$ voltdb compile -o newcatalog.jar  myschema.sql
$ voltadmin update --host=voltsvr3 newcatalog.jar deployment.xml
```

You can use live updates to perform the following actions:

- Add, remove, and update stored procedures
- Add and remove tables
- Add, remove, and in many cases change the datatype of individual columns in a table
- Add, remove, and update indexes

The only limitation on indexes is that you cannot add or broaden a unique index, since there may already be content within the database that would violate the new constraint.

Two other limitations on live schema updates are that you cannot rename tables or columns and you cannot change the partitioning of tables. That is, you cannot change a partitioned table to replicated, a replicated table to partitioned, or change the partitioning column of an existing partitioned table. However, you can change partitioning using the save and restore commands, as described in the following section.

4.2.2. Performing Updates Using Save and Restore

If you need to add unique indexes, add columns to an existing unique index, or change the partitioning of your database tables, you must use the **voltadmin save** and **restore** commands to perform the schema update. This requires shutting down and restarting the database to allow VoltDB to repartition the data and validate any new constraints.

To perform a schema update using save and restore, use the following steps:

1. Compile a new application catalog containing the updated schema and stored procedures.
2. Pause the database (**voltadmin pause**).
3. Create a snapshot of the database contents (**voltadmin save**).
4. Shutdown the database (**voltadmin shutdown**).

5. Create a new database using the **voltdb create** option, the newly recompiled catalog, and starting in admin mode (specified in the deployment file).

6. Restore the snapshot created in Step #3 (**voltadmin restore**).

7. Return the database to normal operations (**voltadmin resume**).

For example:

```
$ voltdb compile -o newcatalog.jar  myschema.sql
$ voltadmin pause
$ voltadmin save --blocking /opt/archive/  mydb
$ voltadmin shutdown

$ # Issue next command on all servers
$ voltdb create newcatalog.jar \
                --deployment=deployment.xml \
                --host=voltsvr1 \
                --license=~/license.xml

$ # Issue only once
$ voltadmin restore /opt/archive  mydb
$ voltadmin resume
```

The key point to remember when changing unique indexes is that if you add new constraints, there is the possibility that the restore operation will fail if existing records violate the new constraint. This is why it is important to make sure your database contents are compatible with the new schema before performing the update.

4.3. Upgrading the Cluster

Sometimes you need to update or reconfigure the server infrastructure on which the VoltDB database is running. Server upgrades are one example. A *server upgrade* is when you need to fix or replace hardware, update the operating system, or otherwise modify the underlying system. Server upgrades usually require stopping the VoltDB database process on the specific server being serviced.

Another example is when you want to reconfigure the cluster as a whole. Reasons for reconfiguring the cluster are because you want to add or remove servers from the cluster or you need to modify the number of partitions per server that VoltDB uses.

Adding servers to the cluster can happen without stopping the database. This is called *elastic scaling*. Removing servers or changing the number of sites per host requires restarting the cluster during a *maintenance window*.

The following sections describe three cases of cluster upgrades:

- Performing server upgrades
- Adding servers to a running cluster through elastic scaling
- Reconfiguring the cluster with a maintenance window

4.3.1. Performing Server Upgrades

If you need to upgrade or replace the hardware or software (such as the operating system) of the individual servers, this can be done without taking down the database as a whole. As long as the server is running with a K-safety value of one or more, it is possible to take a server out of the cluster without stopping the database. You can then fix the server hardware, upgrade software (other than VoltDB), even replace the server entirely with a new server, then bring the server back into the cluster.

To perform a server upgrade:

1. Stop the VoltDB server process on the server. As long as the cluster is K-safe, the rest of the cluster will continue running.
2. Perform the necessary upgrades.
3. Have the server rejoin the cluster using the **voltdb rejoin** command.

The rejoin command starts the database process on the server, contacts the database cluster, then copies the necessary partition content from other cluster nodes so the server can then participate as a full member of the cluster, While the server is rejoining, the other database servers remain accessible and actively process queries from client applications.

When rejoining a cluster you must specify a host server that the rejoining node will connect to. The host can be any server still in the cluster; it does not have to be the same host specified when the cluster was initially started. For example:

```
$ voltdb rejoin --host=voltsvr4 \
         --deployment=deployment.xml \
         --license=~/license.xml
```

Note that you do not need to specify the application catalog. It is downloaded from the other cluster nodes as part of the rejoin operation.

If you need to upgrade all of the servers in the cluster (for example, if you are upgrading the operating system), the easiest method is to upgrade the servers one at a time, taking each server out of the cluster, upgrading it, then rejoining it to the cluster. This way the entire cluster can be upgraded without losing any availability to the database.

If the cluster is not K-safe — that is, the K-safety value is 0 — then you must follow the instructions in Section 4.3.3, “Reconfiguring the Cluster During a Maintenance Window” to upgrade the servers.

4.3.2. Adding Servers to a Running Cluster with Elastic Scaling

If you want to add servers to a VoltDB cluster — usually to increase performance and/or capacity — you can do this without having to restart the database. You add servers to the cluster with the **voltdb add** command, specifying one of the existing nodes with the `--host` flag. For example:

```
$ voltdb add --host=voltsvr4 \
         --license=~/license.xml
```

You must add a full complement of servers to match the K-safety value (K+1) before the servers can participate in the cluster. For example, if the K-safety value is 2, you must add 3 servers before they actually become part of the cluster and the cluster rebalances its partitions.

When you add servers to a VoltDB database, the cluster performs the following actions:

1. The new servers are added to the cluster configuration and sent copies of the application catalog and deployment file.

2. Once sufficient servers are added, copies of all replicated tables and their share of the partitioned tables are sent to the new servers.

3. As the data is rebalanced, the new servers begin processing transactions for the partition content they have received.

4. Once rebalancing is complete, the new servers are full members of the cluster.

4.3.3. Reconfiguring the Cluster During a Maintenance Window

If you want to remove servers from the cluster permanently (as opposed to temporarily removing them for maintenance as described in Section 4.3, "Upgrading the Cluster") or you want to change other cluster-wide attributes, such as the number of partitions per server, you need to restart the server. Stopping the database temporarily to perform this sort of reconfiguration is known as a *maintenance window*.

The steps for reconfiguring the cluster with a maintenance window are:

1. Place the database in admin mode (**voltadmin pause**).

2. Perform a manual snapshot of the database (**voltadmin save**).

3. Shutdown the database (**voltadmin shutdown**).

4. Make the necessary changes to the deployment file.

5. Start a new database using the **voltdb create** option, the existing catalog, and the edited deployment file.

6. Restore the snapshot created in Step #2 (**voltadmin restore**).

7. Return the database to normal operations (**voltadmin resume**).

4.4. Upgrading VoltDB Software

Finally, as new versions of VoltDB become available, you will want to upgrade the VoltDB software on your database cluster. New versions of VoltDB often introduce enhancements to the structure of the application catalog and the execution plans that the database engine uses. To avoid incompatibilities between versions, you should always recompile your application catalog when upgrading.

The steps for upgrading the VotlDB software on a database cluster are:

1. Place the database in admin mode (**voltadmin pause**).

2. Perform a manual snapshot of the database (**voltadmin save**).

3. Shutdown the database (**voltadmin shutdown**).

4. Upgrade VoltDB on all cluster nodes.

5. Recompile the application catalog using the new version of VoltDB.

6. Start a new database using the **voltdb create** option, the newly recompiled catalog, and starting in admin mode (specified in the deployment file).

7. Restore the snapshot created in Step #2 (**voltadmin restore**).

8. Return the database to normal operations (**voltadmin resume**).

Chapter 5. Monitoring VoltDB Databases

Monitoring is an important aspect of systems administration. This is true of both databases and the infrastructure they run on. The goals for database monitoring include ensuring the database meets its expected performance target as well as identifying and resolving any unexpected changes or infrastructure events (such as server failure or network outage) that can impact the database. This chapter explains:

- How to monitor overall database health and performance using VoltDB
- How to integrate VoltDB monitoring with other enterprise monitoring infrastructures

5.1. Monitoring Overall Database Activity

VoltDB provides several tools for monitoring overall database activity. The following sections describe the three main monitoring tools within VoltDB:

- Web Studio
- Process Memory Report
- System Procedures

A separate book, the *VoltDB Enterprise Manager Guide*, describes the VoltDB Enterprise Manager, a complete database management dashboard that includes monitoring of its own.

5.1.1. Web Studio

```
http://voltserver:8080/studio/
```

Web Studio provides a graphical display of key aspects of database performance, including throughput, memory usage, query latency, and partition usage. To use Web Studio, connect to one of the cluster nodes using a web browser, specifying the httpd port (8080 by default) and the page /studio/, as shown in the example URL above. Once the studio page loads, click on the icon of a graph in the menu bar to bring up the monitor display. The monitor shows two graphs, side by side. You can use the pull down menus above each graph to switch between the four available metrics.

The web studio graphs update automatically. Note that Web Studio also lets you issue ad hoc queries to the database and review the database schema.

5.1.2. Process Memory Report

```
http://voltserver:8080/memory/
```

The Process Memory Report is another graphical display focused specifically on memory usage. The report separates out Java heap from non-Java memory usage, so you can better understand what aspects of memory utilization are being exercised. To view the process memory report for a specific VoltDB server, connect to the server using a web browser, specifying the httpd port (8080 by default) and the page /memory/, as shown in the example URL above. Use the browser's refresh button to update the graph.

Note that the process memory report is server-specific, unlike the web studio monitor which shows statistics for the cluster as a whole. If you think you have memory issues with specific servers — particularly if the servers are not all the same size or your data is not evenly distributed across partitions — it is a good idea to examine the process memory report for all of the servers to determine where the issue lies.

5.1.3. System Procedures

VoltDB provides callable system procedures that return detailed information about the usage and performance of the database. In particular, the @Statistics system procedure provides a wide variety of information depending on the selector keyword you give it. Some selectors that are particularly useful for monitoring include the following:

- **MEMORY** — Provides statistics about memory usage for each node in the cluster. Information includes the resident set size (RSS) for the server process, the Java heap size, heap usage, available heap memory, and more. This selector provides the type of information displayed by the Process Memory Report, except that it returns information for all nodes of the cluster in a single call.

- **PROCEDUREPROFILE** — Summarizes the performance of individual stored procedures. Information includes the minimum, maximum, and average execution time as well as the number of invocations, failures, and so on. The information is summarized from across the cluster as whole. This selector returns information similar to the latency graph in Web Studio

- **TABLE** — Provides information about the size, in number of tuples and amount of memory consumed, for each table in the database. The information is segmented by server and partition, so you can use it to report the total size of the database contents or to evaluate the relative distribution of data across the servers in the cluster.

When using the @Statistics system procedure with the PROCEDUREPROFILE selector for monitoring, it is a good idea to set the second parameter of the call to "1" so each call returns information since the last call. In other words, statistics for the interval since the last call. Otherwise, if the second parameter is "0", the procedure returns information since the database started and the aggregate results for minimum, maximum, and average execution time will have little meaning.

When calling @Statistics with the MEMORY or TABLE selectors, you can set the second parameter to "0" since the results are always a snapshot of the memory usage and table volume at the time of the call. For example, the following Python script uses @Statistics with the MEMORY and PROCEDUREPROFILE selectors to check for memory usage and latency exceeding certain limits. Note that the call to @Statistics uses a second parameter of 1 for the PROCEDUREPROFILE call and a parameter value of 0 for the MEMORY call.

```
import sys
from voltdbclient import *

nano = 1000000000.0
memorytrigger = 4 * (1024*1024)     # 4gbytes
avglatencytrigger = .01 * nano      # 10 milliseconds
maxlatencytrigger = 2 * nano        # 2 seconds

server = "localhost"
if (len(sys.argv) > 1): server = sys.argv[1]

client = FastSerializer(server, 21212)
stats = VoltProcedure( client, "@Statistics",
   [ FastSerializer.VOLTTYPE_STRING,
```

```
     FastSerializer.VOLTTYPE_INTEGER ] )

# Check memory
response = stats.call([ "memory", 0 ])
for t in response.tables:
   for row in t.tuples:
      print 'RSS for node ' + row[2] + "=" + str(row[3])
      if (row[3] > memorytrigger):
         print "WARNING: memory usage exceeds limit."

# Check latency
response = stats.call([ "procedureprofile", 1 ])
avglatency = 0
maxlatency = 0
for t in response.tables:
   for row in t.tuples:
      if (avglatency < row[4]): avglatency = row[4]
      if (maxlatency < row[6]): maxlatency = row[6]
print 'Average latency= ' + str(avglatency)
print 'Maximum latency= ' + str(maxlatency)
if (avglatency > avglatencytrigger):
   print "WARNING: Average latency exceeds limit."
if (maxlatency > maxlatencytrigger):
   print "WARNING: Maximum latency exceeds limit."

client.close()
```

The @Statistics system procedure is the the source for many of the monitoring options discussed in this chapter. Two other system procedures, @SystemCatalog and @SystemInformation, provide general information about the database schema and cluster configuration respectively and can be used in monitoring as well.

The system procedures are useful for monitoring because they let you customize your reporting to whatever level of detail you wish. The other advantage is that you can automate the monitoring through scripts or client applications that call the system procedures. The downside, of course, is that you must design and create such scripts yourself. As an alternative for custom monitoring, you can consider integrating VoltDB with existing third party monitoring applications, as described in next section.

5.2. Integrating VoltDB with Other Monitoring Systems

In addition to the tools and system procedures that VoltDB provides for monitoring the health of your database, you can also integrate this data into third-party monitoring solutions so they become part of your overall enterprise monitoring architecture. VoltDB supports integrating VoltDB statistics and status with the following monitoring systems:

- Ganglia
- JMX
- Nagios
- New Relic

5.2.1. Integrating with Ganglia

If you use Ganglia as your monitoring tool and the VoltDB Enterprise Manager as your database administration tool, the two products integrate seamlessly to provide VoltDB performance data to the Ganglia monitoring interface. Ganglia is a distributed monitoring system that provides a graphical interface to distributed clusters of systems. If Ganglia is present, VoltDB and the Enterprise Manager act as a data source for the Ganglia system.

To use VoltDB with Ganglia, make sure:

- The Ganglia Monitoring Daemon (gmond) is installed and configured on each VoltDB cluster node.
- The Ganglia Meta Daemon (gmetad) is installed and configured on the server running the VoltDB Enterprise Manager.

Having completed these steps, the VoltDB Enterprise Manager will automatically generate data for the Ganglia monitoring system. See the Ganglia web site (`http://ganglia.sourceforge.net/`) for more information about Ganglia.

5.2.2. Integrating Through JMX

VoltDB Enterprise Edition uses the Java Management Extensions (JMX) to send statistics from the database nodes to the VoltDB Enterprise Manager. The VoltDB JMX interface is also available to other monitoring frameworks that want to query for VoltDB statistics.

VoltDB Enterprise Edition servers open the JMX interface on port 9090 by default (you can change the port number, see Section A.5.6, “JMX Port” for details). Clients can either poll on this port for specific information or subscribe to messages that are sent approximately every second.

The information sent over the JMX interface is the same as that available through existing VoltDB system procedures, such as @SystemInformation, @Statitstics, and @SnapshotStatus. The difference is that the JMX interface is a lightweight process and is not transactional, as system procedures are. Therefore the JMX interface produces less load on the servers than repeated calls to the system procedures would.

The easiest way to become familiar with the JMX interface is to connect to a running database using the Java Monitoring and Management Console (also known as Jconsole) and browse through the structures returned by the VoltDB servers.

5.2.3. Integrating with Nagios

If you use Nagios to monitor your systems and services, you can include VoltDB in your monitoring infrastructure. VoltDB Enterprise Edition provides Nagios plugins that let you monitor four key aspects of VoltDB. The plugins are included in a subfolder of the tools directory where VoltDB is installed. Table 5.1, “Nagios Plugins” lists each plugin and what it monitors.

Table 5.1. Nagios Plugins

Plugin	Monitors	Scope	Description
check_voltdb_ports	Availability	Server	Reports whether the specified server is reachable or not.
check_voltdb_memory	Memory usage	Server	Reports on the amount of memory in use by VoltDB for a individual node. You can specify the severity criteria as a percentage of total memory.

Plugin	Monitors	Scope	Description
check_voltdb_cluster	K-safety	Cluster-wide	Reports on whether a K-safe cluster is complete or not. That is, whether the cluster has the full complement of nodes or if any have failed and not rejoined yet.
check_voltdb_replication	Database replication	Cluster-wide	Reports the status of database replication. Connect the plugin to one or more nodes on the master database.

Note that the httpd and JSON options must be enabled in the deployment file for the VoltDB database for the Nagios plugins to query the database status.

5.2.4. Integrating with New Relic

If you use New Relic as your monitoring tool, there is a VoltDB plugin to include monitoring of VoltDB databases to your New Relic dashboard. To use the New Relic plugin, you must:

- Define the appropriate configuration for your server.
- Start the voltdb-newrelic process that gathers and sends data to New Relic.

You define the configuration by editing and renaming the template files that can be found in the `/tools/monitoring/newrelic/config` folder where VoltDB is installed. The configuration files let you specify your New Relic license and which databases are monitored. A README file in the `/newrelic` folder provides details on what changes to make to the configuration files.

You start the monitoring process by running the script `voltdb-newrelic` that also can be found in the `/newrelic` folder. The script must be running for New Relic to monitor your databases.

Chapter 6. What to Do When Problems Arise

As with any high performance application, events related to the database process, the operating system, and the network environment can impact how well or poorly VoltDB performs. When faced with performance issues, or outright failures, the most important task is identifying and resolving the root cause. VoltDB and the server produce a number of log files and other artifacts that can help you in the diagnosis. This chapter explains:

- Where to look for log files and other information about the VoltDB server process
- How to collect the log files and other system information when reporting a problem to VoltDB

6.1. Where to Look for Answers

The first place to look when an unrecognized problem occurs with your VoltDB database is the console where the database process was started. VoltDB echoes key messages and errors to the console. For example, if a server becomes unreachable, the other servers in the cluster will report an error indicating which node has failed. Assuming the cluster is K-safe, the remaining nodes will then re-establish a quorum and continue, logging this event to the console as well.

However, not all messages are echoed on the console.[1] A more complete record of errors, warnings, and informational messages is written to a log file, `log/volt.log`, in a subfolder of the working directory where the VoltDB server process was started. The `volt.log` file can be extremely helpful for identifying unexpected but non-fatal events that occurred earlier and may identify the cause of the current issue.

If VoltDB encounters a fatal error and exits, shutting down the database process, it also attempts to write out a crash file in the current working directory. The crash file name has the prefix "voltdb_crash" followed by a timestamp identifying when the file is created. Again, this file can be useful in diagnosing exactly what caused the crash, since it includes the last error message, a brief profile of the server and a dump of the Java threads running in the server process before it crashed.

To summarize, when looking for information to help analyze system problems, three places to look are:

1. The console where the server process was started.
2. The log file in `log/volt.log`
3. The crash file named `voltdb_crash{timestamp}.txt` in the server process's working directory

6.2. Collecting the Log Files

VoltDB includes a utility that collects all of the pertinent logs for a given server. The log collector retrieves the necessary system and process files from the server, creates a compressed archive file and, optionally, uploads it via SFTP to a support site. For customers requesting support from VoltDB, your support contact will often provide instructions on how and when to use the log collector and where to submit the files.

Note that the database does not need to be running to use the log collector. It can find and collect the log files based solely on the location of the VoltDB root directory where the database was run.

[1] Note that you can change which messages are echoed to the console and which are logged by modifying the Log4j configuration file. See the chapter on logging in the *Using VoltDB* manual for details.

You can run the log collector from the command line, from within the VoltDB Enterprise Manager, or programmatically through the REST interface (for databases started through REST or the Enterprise Manager). The following sections describe how to run the log collector using the three different environments.

6.2.1. Collecting Log Files Using the Command Line

To collect the log files from the command line, use the **voltdb collect** command:

```
$ voltdb collect --prefix=mylogs /home/db/voltdbroot
```

When you run the command you must specify the location of the root directory for the database as an argument to the command. For example, if you are in the same working directory where the database was originally started, by default the root directory is the subdirectory `voltdbroot`.

```
$ voltdb collect --prefix=mylogs $(pwd)/voltdbroot
```

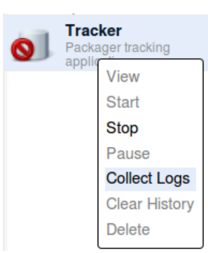
The archive file that the **collect** command generates is created in your current working directory.

The **collect** command has optional arguments that let you control what data is collected, the name of the resulting archive file, as well as whether to upload the file to an FTP server. In the preceding example the `--prefix` flag specifies the prefix for the archive file name. If you are submitting the log files to an FTP server via SFTP, you can use the `--upload`, `--username`, and `--password` flags to identify the target server and account. For example:

```
$ voltdb collect --prefix=mylogs \
     --upload=ftp.mycompany.com \
     --username=babbage
     --password=charles  /home/db/voltdbroot
```

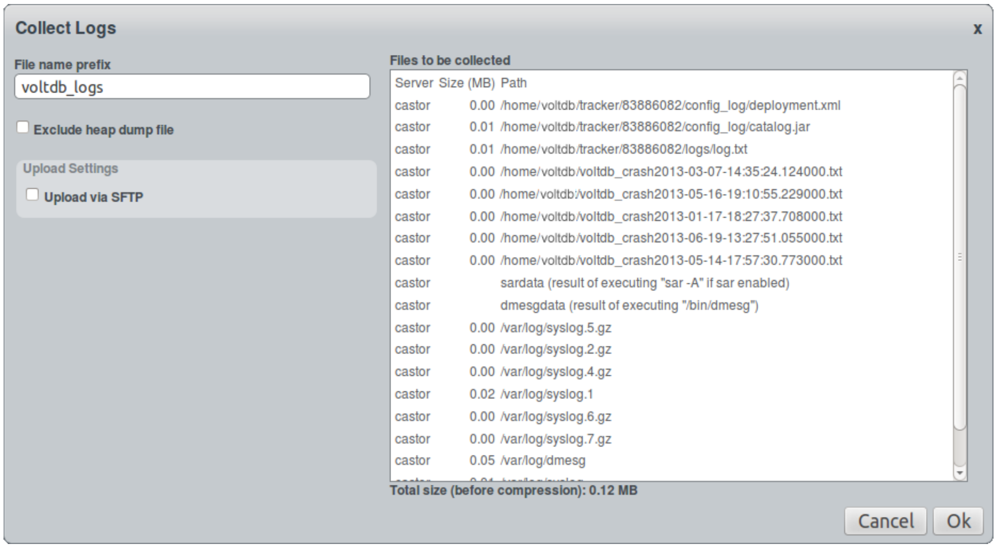
Note that the **voltdb collect** command collects log files for the current system only. To collect logs for all servers in a cluster, you will need to issue the **voltdb collect** command locally on each server separately. See the **voltdb collect** documentation in the *Using VoltDB* manual for details.

6.2.2. Collecting Log Files Using the Enterprise Manager



To collect the log files from a database cluster created and managed by the VoltDB Enterprise Manager, it is easiest to use the Enterprise Manager interface. Click on the name of the database on the database list to the left of the dashboard then select the item "Collect logs" from the popup menu.

This menu item brings up a dialog box showing a list of the files and command output that will be collected. The dialog box also has fields so you can control the resulting file prefix, include or exclude the heap dump file, and optionally upload the results via SFTP.



The major difference between using the Enterprise Manager interface and the shell command line is that the Enterprise Manager creates (and optionally uploads) archives for every server in the cluster at once. You do not need to collect logs for each server separately.

6.2.3. Collecting Log Files Using the REST Interface

Finally, it is possible to invoke the log collector using the REST interface. Invoking the log collector from REST has the same effect as invoking it from within the Enterprise Manager — it collects the logs on all of the servers of the cluster.

The method for collecting logs using the REST interface is `/mgmt/databases/{id}/collect`. Attributes of the method let you optionally customize the collection in the same way you can from the command line or the Enterprise Manager interface. For example, the following REST call collects the logs from all of the servers associated with the database ID 12345, using the file prefix "MyLogs":

```
http://voltdbmgr:9000/man/api/1.0/mgmt
/databases/12345/collect?prefix=MyLogs
```

See the REST reference appendix in the *VoltDB Enterprise Manager Guide* for more information about the collect method and the supported attributes.

Appendix A. Server Configuration Options

There are a number of system, process, and application options that can impact the performance or behavior of your VoltDB database. You control these options when starting VoltDB. The configuration options fall into five main categories:

- Server configuration
- Process configuration
- Database configuration
- Path configuration
- Network ports used by the database cluster

This appendix describes each of the configuration options, how to set them, and their impact on the resulting VoltDB database and application environment.

A.1. Server Configuration Options

VoltDB provides mechanisms for setting a number of options. However, it also relies on the base operating system and network infrastructure for many of its core functions. There are operating system configuration options that you can adjust to to maximize your performance and reliability, including:

- Network configuration
- Time configuration

A.1.1. Network Configuration (DNS)

VoltDB creates a network mesh among the database cluster nodes. To do that, all nodes must be able to resolve the IP address and hostnames of the other server nodes. Make sure all nodes of the cluster have valid DNS entries or entries in the local hosts files.

For servers that have two or more network interfaces — and consequently two or more IP addresses — it is possible to assign different functions to each interface. VoltDB defines two sets of ports:

- External ports, including the client and admin ports. These are the ports used by external applications to connect to and communicate with the database.
- Internal ports, including all other ports. These are the ports used by the database nodes to communicate among themselves. These include the internal port, the zookeeper port, and so on. (See Section A.5, "Network Ports" for a complete listing of ports.)

You can specify which network interface the server expects to use for each set of ports by specifying the internal and external interface when starting the database. For example:

```
$ voltdb create catalog.jar -d deployment.xml \
  -H serverA -l license.xml \
  --externalinterface=10.11.169.10 \
  --internalinterface=10.12.171.14
```

A.1.2. Time Configuration (NTP)

Keeping VoltDB cluster nodes in close synchronization is important for the ongoing performance of your database. At a minimum, use of NTP to synchronize time across the cluster is recommended. If the time difference between nodes is too large (greater than three seconds) VoltDB refuses to start. It is also important to avoid having nodes adjust time backwards, or VoltDB will pause while it waits for time to "catch up" to its previous setting.

A.2. Process Configuration Options

In addition to system settings, there are configuration options pertaining to the VoltDB server process itself that can impact performance. Runtime configuration options are set as command line options when starting the VoltDB server process.

The key process configuration for VoltDB is the Java maximum heap size. It is also possible to pass other arguments to the Java Virtual Machine directly.

A.2.1. Maximum Heap Size

The heap size is a parameter associated with the Java runtime environment. Certain portions of the VoltDB server software use the Java heap. In particular, the part of the server that receives and responds to stored procedure requests uses the Java heap.

Depending upon how many transactions your application executes a second, you may need additional heap space. The higher the throughput, the larger the maximum heap needed to avoid running out of memory.

In general, a maximum heap size of two gigabytes (2048) is recommended. For production use, a more accurate measurement of the needed heap size can be calculated from the size of the catalog (number of tables), number of sites per host, and what durability and availability features are in use. See the *VoltDB Planning Guide* for details.

It is important to remember that the heap size is not directly related to data storage capacity. Increasing the maximum heap size does not provide additional data storage space. In fact, quite the opposite. Needlessly increasing the maximum heap size reduces the amount of memory available for storage.

To set the maximum heap size when starting VoltDB, define the environment variable VOLTDB_HEAPMAX as an integer value (in megabytes) before issuing the **voltdb** command. For example, the following commands start VoltDB with a 3 gigabyte heap size (the default is 2 gigabytes):

```
$ export VOLTDB_HEAPMAX="3072"
$ voltdb create mycatalog.jar \
   -d deployment.xml -H serverA
```

A.2.2. Other Java Runtime Options (VOLTDB_OPTS)

VoltDB sets the Java options — such as heap size and classpath — that directly impact VoltDB. There are a number of other configuration options available in the Java Virtual machine (JVM).

VoltDB provides a mechanism for passing arbitrary options directly to the JVM. If the environment variable VOLTDB_OPTS is defined, its value is passed as arguments to the Java command line.

- When starting VoltDB using the **voltdb** command, the contents of VOLTDB_OPTS are added to the Java command line on the current server.

- When starting the VoltDB Enterprise Manager, the contents of VOLTDB_OPTS are added to the invocation that the Enterprise Manager uses to start the VoltDB process on all remote servers.

In other words, when starting a cluster manually, you must define VOLTDB_OPTS on each server to have it take effect for all servers. When using the Enterprise Manager, you only need to define VOLTDB_OPTS once, before starting the Enterprise Manager, to have these JVM options take effect on all nodes that the Enterprise Manager starts.

Warning

VoltDB does not validate the correctness of the arguments you specify using VOLTDB_OPTS or their appropriateness for use with VoltDB. This feature is intended for experienced users only and should be used with extreme caution.

A.3. Database Configuration Options

Runtime configuration options are set either as part of the deployment file or as command line options when starting the VoltDB server process. These database configuration options are only summarized here. See the *Using VoltDB* manual for a more detailed explanation. The configuration options include:

- Sites per host
- K-Safety
- Network partition detection
- Automated snapshots
- Export
- Command logging
- Heartbeat
- Temp table size

A.3.1. Sites per Host

Sites per host specifies the number of unique VoltDB "sites" that are created on each physical database server. The section on "Determining How Many Partitions to Use" in the *Using VoltDB* manual explains how to choose a value for sites per host.

You set the value of sites per host using the `sitesperhost` attribute of the `<cluster>` tag in the deployment file.

A.3.2. K-Safety

K-safety defines the level of availability or durability that the database can sustain, by replicating individual partitions to multiple servers. K-safety is described in detail in the "Availability" chapter of the *Using VoltDB* manual.

You specify the level of K-safety that you want in the deployment file using the `kfactor` attribute of the `<cluster>` tag.

A.3.3. Network Partition Detection

Network partition detection protects a VoltDB cluster in environments where the network is susceptible to partial or intermittent failure among the server nodes. Partition detection is described in detail in the "Availability" chapter of the *Using VoltDB* manual.

Use of network partition detection is strongly recommended for production systems and therefore is enabled by default. You can enable or disable network partition detection in the deployment file using the <partition-detection> tag.

A.3.4. Automated Snapshots

Automated snapshots provide ongoing protection against possible database failure (due to hardware or software issues) by taking periodic snapshots of the database's contents. Automated snapshots are described in detail in the section on "Scheduling Automated Snapshots" in the *Using VoltDB* manual.

You enable and configure automated snapshots with the <snapshot> tag in the deployment file.

Snapshot activity involves both processing and disk I/O and so may have a noticeable impact on performance (in terms of throughput and/or latency) on a very busy database. You can control the priority of snapshots activity using the <snapshot/> tag within the <systemsettings> element of the deployment file. The snapshot priority is an integer value between 0 and 10, with 0 being the highest priority and 10 being the lowest. The closer to 10, the longer snapshots take to complete, but the less they can affect ongoing database work.

Note that snapshot priority affects all snapshot activity, including automated snapshots, manual snapshots, and command logging snapshots.

A.3.5. Export

The export function lets you automatically export selected data from your VoltDB database to an export client using SQL INSERT statements to special export tables at runtime. This feature is described in detail in the chapter on "Exporting Live Data" in the *Using VoltDB* manual.

You enable and disable export using the <export> tag in the deployment file.

A.3.6. Command Logging

The command logging function saves a record of each transaction as it is initiated. These logs can then be "replayed" to recreate the database's last known state in case of intentional or accidental shutdown. This feature is described in detail in the chapter on "Command Logging and Recovery" in the *Using VoltDB* manual.

To enable and disable command logging, use the <commandlog> tag in the deployment file.

A.3.7. Heartbeat

The database servers use a "heartbeat" to verify the presence of other nodes in the cluster. If a heartbeat is not received within a specified time limit, that server is assumed to be down and the cluster reconfigures itself with the remaining nodes (assuming it is running with K-safety). This time limit is called the "heartbeat timeout" and is specified as a integer number of seconds.

For most situations, the default value for the timeout (10 seconds) is appropriate. However, if your cluster is operating in an environment that is susceptible to network fluctuations or unpredictable latency, you may want to increase the heartbeat timeout period.

You can set an alternate heartbeat timeout using the `<heartbeat>` tag in the deployment file.

A.3.8. Temp Table Size

VoltDB uses temporary tables to store intermediate table data while processing transactions. The default temp table size is 100 megabytes. This setting is appropriate for most applications. However, extremely complex queries or many updates to large records could cause the temporary space to exceed the maximum size, resulting in the transaction failing with an error.

In these unusual cases, you may need to increase the temp table size. You can specify a different size for the temp tables using the `<temptables>` tag in the deployment file. Note: since the temp tables are allocated as needed, increasing the maximum size can result in a Java out-of-memory error at runtime if the system is memory-constrained. Modifying the temp table size should be done with caution.

A.4. Path Configuration Options

The running database uses a number of disk locations to store information associated with runtime features, such as export, network partition detection, and snapshots. You can control which paths are used for these disk-based activities. The path configuration options include:

- VoltDB root
- Snapshots path
- Export overflow path
- Command log path
- Command log snapshots path

A.4.1. VoltDB Root

VoltDB defines a root directory for any disk-based activity which is required at runtime. This directory also serves as a root for all other path definitions that take the default or use a relative path specification.

By default, the VoltDB root is the directory `voltdbroot` created as a subfolder in the current working directory for the process that starts the VoltDB server process. (If the subfolder does not exist, VoltDB creates it on startup.) You can specify an alternate root in the deployment file using the `<voltdbroot>` element. However, if you explicitly name the root directory in the deployment file, the directory must exist or the database server cannot start. See the section on "Configuring Paths for Runtime Features" in the *Using VoltDB* manual for details.

When using the VoltDB Enterprise Manager, the VoltDB root directory is defined implicitly by the destination directory. You define the destination directory when you create the database. The VoltDB root becomes a subfolder of the destination directory, where the subfolder name is the same as the database ID of the current database. So, for example, if the destination directory is `/opt/voltdb` and the database ID is 12345, the resulting VoltDB root directory is `/opt/voltdb/12345`.

A.4.2. Snapshots Path

The snapshots path specifies where automated and network partition snapshots are stored. The default snapshots path is the "`snapshots`" subfolder of the VoltDB root directory. You can specify an alternate path for snapshots using the `<snapshots>` child element of the `<paths>` tag in the deployment file.

A.4.3. Export Overflow Path

The export overflow path specifies where overflow data is stored if the export queue gets too large. The default export overflow path is the "`export_overflow`" subfolder of the VoltDB root directory. You can specify an alternate path using the `<exportoverflow>` child element of the `<paths>` tag in the deployment file.

See the chapter on "Exporting Live Data" in the *Using VoltDB* manual for more information on export overflow.

A.4.4. Command Log Path

The command log path specifies where the command logs are stored when command logging is enabled. The default command log path is the "`command_log`" subfolder of the VoltDB root directory. However, for production use, it is strongly recommended that the command logs be written to a dedicated device, not the same device used for snapshotting or export overflow. You can specify an alternate path using the `<commandlog>` child element of the `<paths>` tag in the deployment file.

See the chapter on "Command Logging and Recovery" in the *Using VoltDB* manual for more information on command logging.

A.4.5. Command Log Snapshots Path

The command log snapshots path specifies where the snapshots created by command logging are stored. The default path is the "`command_log_snapshot`" subfolder of the VoltDB root directory. (Note that command log snapshots are stored separately from automated snapshots.) You can specify an alternate path using the `<commandlogsnapshot>` child element of the `<paths>` tag in the deployment file.

See the chapter on "Command Logging and Recovery" in the *Using VoltDB* manual for more information on command logging.

A.5. Network Ports

A VoltDB cluster opens network ports to manage its own operation and to provide services to client applications. When using the Enterprise Manager, most ports are configurable as part of the database definition. Many of the network ports are configurable as part of the command line that starts the VoltDB database process or through the deployment file. Table A.1, “VoltDB Port Usage” summarizes the ports that VoltDB uses, their default value, and how to change the default. The following sections describe each port in more detail.

Table A.1. VoltDB Port Usage

Port	Default Value	Where to Set (Community Edition)
Client Port	21212	VoltDB command line
Admin Port	21211	Deployment file
Web Interface Port (httpd)	8080	Deployment file
Internal Server Port	3021	VoltDB command line
Log Port	4560	(Enterprise Manager only)
JMX Port	9090	VoltDB command line
Replication Port	5555	VoltDB command line

Port	Default Value	Where to Set (Community Edition)
Zookeeper port	2181	VoltDB command line

A.5.1. Client Port

The client port is the port VoltDB client applications use to communicate with the database cluster nodes. By default, VoltDB uses port 21212 as the client port. You can change the client port. However, all client applications must then use the specified port when creating connections to the cluster nodes.

To specify a different client port on the command line, use the `--client` when starting the VoltDB database. For example, the following command starts the database using port 12345 as the client port:

```
$ voltdb create mycatalog.jar -l ~/license.xml \
         -d deployment.xml -H serverA \
         --client=12345
```

When using the Enterprise Manager, use the Edit Configuration dialog box to specify the port to use.

If you change the default client port, all client applications must also connect to the new port. The client interfaces for Java and C++ accept an additional, optional argument to the createConnection method for this purpose. The following examples demonstrate how to connect to an alternate port using the Java and C++ client interfaces.

Java

```
org.voltdb.client.Client voltclient;
voltclient = ClientFactory.createClient();
voltclient.createConnection("myserver",12345);
```

C++

```
boost::shared_ptr<voltdb::Client> client = voltdb::Client::create();
client->createConnection("myserver", 12345);
```

A.5.2. Admin Port

The admin port is similar to the client port, it accepts and processes requests from applications. However, the admin port has the special feature that it continues to accept requests when the database enters admin mode.

By default, VoltDB uses port 21211 as the admin port. You can change the port assignment in the deployment file using the <admin-mode> tag or on the command line using the `--admin` flag. For example, the following deployment file sets the admin port to 2222:

```
<deployment>
   ...
   <admin-mode port="2222" />
</deployment>
```

The same effect can be achieved using the --admin flag on the command line:

```
$ voltdb create mycatalog.jar -l ~/license.xml \
         -d deployment.xml -H serverA \
         --admin=2222
```

When the admin port is set in both the deployment file and on the command line, the command line setting supersedes the deployment file.

When using the Enterprise Manager, use the Edit Configuration dialog box to specify a different admin port.

A.5.3. Web Interface Port (httpd)

The web interface port is the port that VoltDB listens to for web-based connections from the JSON interface. There are two related settings associated with the JSON interface. The first setting is whether the port is enabled; the second is which port to use, if the interface is enabled.

When starting a VoltDB database manually, the web interface port (and the JSON interface) or disabled by default. When using the Enterprise Manager, both the web interface port and the JSON interface are enabled by default. The default httpd port is 8080.

If you plan on using the JSON interface from the community edition, be sure to include the <httpd> tag in the deployment file.

- To enable the httpd port but disable the JSON interface, specify the attribute `enabled="false"` in the `<jsonapi>` tag in the deployment file when starting VoltDB. If you are using the Enterprise Manager, there is a check box for enabling and disabling the JSON interface in the Edit Configuration dialog box.
- To change the web interface port, specify the alternate port using the `port` attribute to the `<httpd>` tag in the deployment file. Or, if you are using the Enterprise Manager use the httpd port field in the Edit Configuration dialog.

For example, the following deployment file fragment enables the web interface and the JSON interface, specifying the alternate port 8083.

```
<httpd port='8083'>
        <jsonapi enabled='true'/>
</httpd>
```

If you change the port number, be sure to use the new port number when connecting to the cluster using the JSON interface. For example, the following URL connects to the port 8083, instead of 8080:

```
http://athena.mycompany.com:8083/api/1.0/?Procedure=@SystemInformation
```

For more information about the JSON interface and specifying the appropriate port when connecting to the VoltDB cluster, see the section on "How the JSON Interface Works" in the *Using VoltDB* manual.

A.5.4. Internal Server Port

A VoltDB cluster uses ports to communicate among the cluster nodes. This port is internal to VoltDB and should not be used by other applications.

By default, the internal server port is port 3021 for all nodes in the cluster[1]. You can specify an alternate port using the `--internal` flag when starting the VoltDB process. For example, the following command starts the VoltDB process using an internal port of 4000:

[1] In the special circumstance where multiple VoltDB processes are started for one database, all on the same server, the internal server port is incremented from the initial value for each process.

```
$ voltdb create mycatalog.jar -l ~/license.xml \
         -d deployment.xml -H serverA \
          --internal=4000
```

A.5.5. Log Port

When using the Enterprise Manager to configure and run VoltDB, the resulting VoltDB cluster nodes open a port as an output stream for log4J messages. The Enterprise Manager uses the port to fetch log4J messages from the cluster nodes and display them in the management console.

By default, port 4560 is assigned as the log port. However, this port is only opened when using the Enterprise Manager. You can change the port number using the Edit Database dialog from within the VoltDB Enterprise Manager.

A.5.6. JMX Port

The VoltDB Enterprise Manager uses JMX to collect statistics from the cluster nodes at runtime. It does this using JMX.

The default JMX port is 9090. However, the JMX port is only opened when using the VoltDB Enterprise Edition. You can change the port number used by JMX by adding the Java argument -Dvolt.rmi.agent.port to the command line. You do that by defining the environment variable VOLTDB_OPTS before starting the server. For example, the following command assigns the JMX port to port number 2345:

```
$ export VOLTDB_OPTS="-Dvolt.rmi.agent.port=2345"
```

There is no JMX port when using the VoltDB Community Edition.

A.5.7. Replication Port

During database replication, the DR agent uses three ports to connect to the master database. By default, these ports are three sequential ports starting at port 5555; in other words, ports 5555, 5556, and 5557. You can use three different ports by specifying a different starting port either on the **voltdb** command line or in the deployment file.

- On the command line, use the `--replication` flag to specify a different starting port:

  ```
  $ voltdb create mycatalog.jar -l ~/license.xml \
           -d deployment.xml -H serverA \
          --replication=6666
  ```

- In the deployment file, specify the starting port number using the <replication> tag:

  ```
  <replication port="6666" />
  ```

Adding the replication port to the deployment file is useful when setting the port for all nodes in the cluster. Using the command line option is useful for changing the default port for only one node in the cluster. If you specify the replication port in both the deployment file and on the command line, the command line argument takes precedence.

A.5.8. Zookeeper Port

VoltDB uses a version of Apache Zookeeper to communicate among supplementary functions that require coordination but are not directly tied to database transactions. Zookeeper provides reliable synchronization for functions such as command logging without interfering with the database's own internal communications.

VoltDB uses a network port bound to the local interface (127.0.0.1) to interact with Zookeeper. By default, 2181 is assigned as the Zookeeper port. You can specify a different port number using the `--zookeeper` flag when starting the VoltDB process. For example:

```
$ voltdb create mycatalog.jar -l ~/license.xml \
         -d deployment.xml -H serverA \
         --zookeeper=2288
```

Appendix B. Snapshot Utilities

VoltDB provides two utilities for managing snapshot files. These utilities verify that a native snapshot is complete and usable and convert the snapshot contents to a text representation that can be useful for uploading or reloading data in case of severe errors.

It is possible, as the result of a design flaw or failed program logic, for a database application to become unusable. However, the data is still of value. In such emergency cases, it is desirable to extract the data from the database and possibly reload it. This is the function that save and restore performs within VoltDB.

But there may be cases when you want to use the data created by a VoltDB snapshot elsewhere. The goal of the utilities is to assist in that process. The snapshot utilities are:

- *SnapshotConverter* converts a snapshot (or part of a snapshot) into text files, creating one file for each table in the snapshot.
- *SnapshotVerifier* verifies that a VoltDB snapshot is complete and usable.

Unlike system procedures, that must be run within the context of an existing database connection, the snapshot utilities can be run from the command line without a running database present. However, the utilities are still dependent on the Java classes and library for VoltDB. So you must be sure to define your Java classpath and library path appropriately to invoke the classes.

To run either SnapshotVerifier or SnapshotConverter, use the Java command to invoke the class, specifying the appropriate classpath and library path based on where your VoltDB software is installed. For example, if VoltDB is installed in /opt/voltdb, the command to invoke SnapshotVerifier is as follows:

```
$ java -classpath "/opt/voltdb/voltdb/*" \
       -Djava.library.path=/opt/voltdb/voltdb \
       org.voltdb.utils.SnapshotVerifier
```

You may find it easier to add an alias to your shell script startup file to abbreviate these commands:

```
$ alias snapshotverify="java -cp \"/opt/voltdb/voltdb/*\"  \
        -Djava.library.path=/opt/voltdb/voltdb \
         org.voltdb.utils.SnapshotVerifier "
$ alias snapshotconvert="java -cp \"/opt/voltdb/voltdb/*\"  \
        -Djava.library.path=/opt/voltdb/voltdb \
        org.voltdb.utils.SnapshotConverter "
```

The following sections describing each command assume these aliases have been defined.

Each command accepts a different set of arguments. Use the `--help` argument to display a list the allowable arguments and qualifiers. For example:

```
$ snapshotverify --help
```

snapshotconvert

snapshotconvert — Converts the tables in a VoltDB snapshot into text files.

Syntax

```
snapshotconvert {snapshot-id} --type {csv|tsv} \
        --table {table} [...] [--dir {directory}]... \
        [--outdir {directory}]

snapshotconvert --help
```

Description

SnapshotConverter converts one or more tables in a valid snapshot into either comma-separated (csv) or tab-separated (tsv) text files, creating one file per table.

Where:

{snapshot-id}	is the unique identifier specified when the snapshot was created. (It is also the name of the .digest file that is part of the snapshot.) You must specify a snapshot ID.
{csv|tsv}	is either "csv" or "tsv" and specifies whether the output file is comma-separated or tab-separated. This argument is also used as the filetype of the output files.
{table}	is the name of the database table that you want to export to text file. You can specify the `--table` argument multiple times to convert multiple tables with a single command.
{directory}	is the directory to search for the snapshot (`--dir`) or where to create the resulting output files (`--outdir`). You can specify the `--dir` argument multiple times to search multiple directories for the snapshot files. Both `--dir` and `--outdir` are optional; they default to the current directory path.

Example

The following command exports two tables from a snapshot of the flight reservation example used in the *Using VoltDB* manual. The utility searches for the snapshot files in the current directory (the default) and creates one file per table in the user's home directory:

```
$ snapshotconvert flightsnap --table CUSTOMER --table RESERVATION \
                  --type csv -- outdir ~/
```

snapshotverify

snapshotverify — Verifies that the contents of one or more snapshot files are complete and usable.

Syntax

snapshotverify [snapshot-id] [--dir {directory}] ...

snapshotverify --help

Description

SnapshotVerifier verifies one or more snapshots in the specified directories.

Where:

[snapshot-id]	is the unique identifier specified when the snapshot was created. (It is also the name of the .digest file that is part of the snapshot.) If you specify a snapshot ID, only snapshots matching that ID are verified. If you do not specify an ID, all snapshots found will be verified.
{directory}	is the directory to search for the snapshot. You can specify the `--dir` argument multiple times to search multiple directories for snapshot files. If you do not specify a directory, the default is to search the current directory.

Examples

The following command verifies all of the snapshots in the current directory:

```
$ snapshotverify
```

This example verifies a snapshot with the unique identifier "flight" in either the directory `/etc/voltdb/save` or `~/mysaves`:

```
$ snapshotverify flight --dir /etc/voltdb/save/ --dir ~/mysaves
```