Release Planning

BigWorld Technology 1.9.1. Released 2008.

Software designed and built in Australia by BigWorld.

Level 3, 431 Glebe Point Road
Glebe NSW 2037, Australia
www.bigworldtech.com

Table of Contents

1. Overview 3
2. Developing a Robust Server 4
    2.1. Development Considerations 4
        2.1.1. Blocking the Main Thread 4
        2.1.2. Ghost Entities and Mailboxes 4
        2.1.3. Validating Client Arguments 5
        2.1.4. Fault Tolerance Considerations 5
        2.1.5. Profiling 6
        2.1.6. Server Logs 7
    2.2. Testing 7
    2.3. Communicating with the BigWorld Support Team 7
3. Cluster Hardware 8
    3.1. Choosing the Number of BigWorld Server Instances 8
    3.2. The Cluster Hardware 8
        3.2.1. General Cluster Machines 8
        3.2.2. Database Machines 9
        3.2.3. Server Tools Machines 9
        3.2.4. Secondary Storage Considerations 9
        3.2.5. Sizing the Cluster 10
    3.3. Cluster Configuration 11
        3.3.1. Linux Distributions 11
        3.3.2. Smoothing load during startup 11
        3.3.3. MySQL Configuration 11
        3.3.4. Network Configuration 12
        3.3.5. BWMachined Version Consistency 13
    3.4. Client Hardware Recommendations 13
        3.4.1. Advanced Terrain Recommendations 14
        3.4.2. Classic Terrain Recommendations 14
4. Monitoring and Maintaining the Cluster 15
    4.1. Day To Day Monitoring 15
5. Communicating with the BigWorld Support Team 16
6. Cluster Deployment Examples 17
    6.1. Cluster Machines Distribution 17

Chapter 1. Overview

This document describes the tasks required to deploy a scalable robust game based on BigWorld Technology. The tasks detailed in this document are based on the conclusions which were taken from multiple deployments for different games. BigWorld considers these tasks to be critical for achieving a successful beta period and game release.

The intended audience of this document is senior product developers, art leads, project coordinators and quality assurance leads.

Chapter 2. Developing a Robust Server

Developing and deploying an MMOG is a complicated task. Developers using BigWorld Technology need to be aware of many issues to help avoid problems when deploying their game. This chapter lists things to take into account during development and testing of a BigWorld game in order to help achieve a successful deployment.

2.1. Development Considerations

2.1.1. Blocking the Main Thread

Developers should be careful not to run blocking operations in the main thread. This applies to the server components which run scripts, including the BaseApp, CellApp and DbMgr (when using class-customised data types). This is required to ensure the process can respond to its peers and manager process in an appropriate time. In an extreme case, a process that pauses for too long may be considered as dead by other server components and stopped via a signal from BWMachineD.

This time period is different for different processes. For example, by default, if a CellApp pauses for more than 3 seconds worth of inactivity, the CellAppMgr will kill that CellApp process (a common cause is because of an infinite loop in script). For BaseApps, by default, if they are non-responsive for longer than 5 seconds, the BaseAppMgr will kill that BaseApp process. These limits are configurable, see the *Server Configuration with* `bw.xml`.

One of the following solutions can be used for large or blocking operations:

1. Separate the calculation into multiple steps and use a timer to call each step.

2. Use asynchronous calls to avoid blocking disk access in the main thread. For example, use the `fetchDataSection`, `fetchEntitiesFromChunks` and `fetchFromChunks` methods of BaseApp's `BigWorld` module.

3. Pre-load any data from disk that cannot be accessed via asynchronous methods. This can be done, for example, in the loading of the personality module or any other module loaded during startup.

4. Implement asynchronous calls using an external process. For example: long blocking database queries could be implemented with an intermediate process which will process the queries and send a callback to the caller once done. Example code can be found at `bigworld/src/server/baseapp/eg_tcpecho.cpp` and `eg_tcpechoserver.py`.

It is recommended to review the server warnings during development and to give extra attention to warnings about loading resources in the main thread.

Please also note that operations which take a short time in a test environment might take more time in a production environment.

More information about blocking the main thread can be found in the Server Programming Guide.

2.1.2. Ghost Entities and Mailboxes

Care needs to be taken to avoid situations where Python script makes assumptions that an entity is a Real Entity when it is possible for it to be a Ghost Entity or even an Entity Mailbox. When referring to other entities from script inside an entity, you should never assume that you have a real entity. It is possible for this entity to be a ghost entity or, in some situations, a mailbox to a remote entity.

Generally speaking, the `self` property will be a real entity but entities passed in as argument or found other ways could be ghost entities or mailboxes. The exception to this is when a method is called on a ghost entity that is not in the entity's def file.

To help identify these problems during development, you should enable the bw.xml option cellApp/treatAllOtherEntitiesAsGhosts and disable the option cellApp/shouldResolveMailBoxes during development. For further details please review Server Programming Guide's Debugging section.

Note

Real entities are authoritative, and you can call defined and undefined methods, get and set attributes that are both defined properties and undefined attributes.
Ghost entities are not authoritative, but you can call remote methods and have read access to properties defined as CELL_PUBLIC.
Mailboxes can only call remote methods, and cannot access any attributes on the real entity, defined or otherwise, with the exception that the entity ID and class name can be accessed.

2.1.3. Validating Client Arguments

You should validate any input coming from a client, as gamers might try to modify the game client and exploit the server by sending invalid data.

Arguments to defined methods that are typed as PYTHON are inherently risky and should not be used at all for <Exposed> methods that the client can call on the server. This is because a PYTHON argument might contain code which will run on the server.

PYTHON arguments are useful when developing game system prototypes, but they should be converted to another data type such as FIXED_DICT prior to production. You can class-customise FIXED_DICT and wrap the values with another Python object.

Another reason for this restriction is the performance cost of a PYTHON argument. A PYTHON argument requires pickling of the data type for every send and receive. This can be much slower than simply reading and writing primitive types, and also takes up more space on the network stream.

2.1.4. Fault Tolerance Considerations

2.1.4.1. Scripting

There are some issues relating to the fault tolerance and disaster recovery mechanisms that game script developers need to be aware of when implementing entities:

- Because an entity is periodically saved via BaseApp archiving to DB as well as BaseApp backups (i.e. not on every property state change), the backup data can represent an outdated copy of an entity. This becomes important in scenarios where an entity is restored due to a CellApp or BaseApp process failure.

  Important events in an base entity's lifetime should manually save the entity's state to the database via the Entity or Base method writeToDB().

- It is possible that when a particular entity is backed up or archived to the database, that it is in the middle of a transaction that involves other entities. There is no guarantee that the other entities in that transaction be archived to the database at the same time as this entity. Due to the way the archiving algorithm is randomised, the time when they are saved to the database may differ by up to twice the configured archive period. It is thus important to have journaling data structures (custom to your game script) of the transaction steps to be performed so that transactions can be resumed or rolled back when the base entity is restored from the database.

- When restoring from the database, restored base entities have __init__() called on them. They should check whether BigWorld.hasStarted is False, which indicates that they have been restored from the database. It is important to note that entities require different initialisation handling for being restored as opposed to when they are created using BigWorld.createBase*(), as they must check data consistency of the base and cell entity state which has already been initialised from backups. The cell entity state can be accessed before re-initialising the cell entity via the `cellData` dictionary attribute.

Restored base entities are also responsible for recreating their associated cell entity. The cell entity attributes (such as `spaceID`, `position` and `direction`, as well as cell properties defined in the entity's definition file) will have been preserved in the entity archival along with the rest of the data.

If space archiving is enabled via `<cellAppMgr/archiveSpaceData>`, spaces will also be restored and their associated space ID will remain the same. If using space archiving, base entities can restore their corresponding cell entities by checking the `spaceID` key in the `cellData` dictionary is non-zero, and calling the `BigWorld.createCellEntity`().

BaseApps will have restored the base entities before the BaseApp personality script callback `onBaseAppReady`() is called, which usually triggers loading of entities and setting up spaces. In the case of restoring from database, it is necessary to prevent the normal start up of the game, i.e. loading entities and spaces, as they will already be present when `onBaseAppReady`() is called. An exception to this are cell-only entities, which will have been lost when the server was shut down, and need to be recreated if necessary to the game design.

- Only persistent entities (i.e. entities with defined persistent properties) will be restored, and only those properties that are marked as persistent will have their values restored from the database. The other property values will be set to their default values.

2.1.4.2. Operations

Here are some guidelines on what should be done from an operations viewpoint:

- Use controlled shutdown and startup (the default when stopping a server in WebConsole). In particular, do not prematurely kill processes during shutdown, as data loss may occur.
- If your game design is such that game state is recreated from script rather than restored from the archived database state, make sure that the <`dbMgr/clearRecoveryData`> `bw.xml` option is set to `false`. This ensures that during startup, archived entities that were present at the most recent controlled shutdown are restored.
- Ensure that there are Reviver processes which will revive the BaseAppMgr, CellAppMgr, DBMgr and LoginApps if any of them fail.
- Ensure that there are enough CellApp and BaseApp processes running in the cluster such that the desired performance can be maintained on isolated process failures. There should be procedures in place for analysing process failures, and operations staff should start a new CellApp or BaseApps on a spare machine while analysing the initial process failure.

2.1.5. Profiling

2.1.5.1. Profiling Script Performance

You should be profiling and optimising the performance of your server scripts using the methods specified in the Server Programming Guide document.

2.1.5.2. Profiling Entity Sizes and Bandwidth Usage

As entities are the main game objects being used by all components of the BigWorld engine, it is important to make sure that your game entities are implemented as efficiently as possible. This includes:

- Minimising persistent properties.
- Ensuring properties have the smallest applicable data type (while considering long term scaling).

- Ensuring properties have the most appropriate data propagation flags assigned.
- Ensuring properties have level-of-detail if appropriate.

2.1.6. Server Logs

Ideally, developers should review and fix every `WARNING`, `ERROR` and `CRITICAL` message in the server logs. For those messages that cannot be fixed, the developer should have a good understanding of what the message means and why they are occurring.

In order to assist in reducing the noise when reviewing production logs, it is useful to remove any development / debugging log messages from game code / script. This should include any non-essential `HACK_MSG`, `DEBUG_MSG` as well as all non crucial `print` statements from entity scripts.

2.1.6.1. Collecting Log Data

A summary of the log data over a long period of time can be generated by running **mlcat.py** `--summary` this allows detecting abnormal behaviour. See the **mlcat.py** `--help` for more information.

2.2. Testing

Bot tests should be run as early and as frequently as possible to ensure your game environment scales as expected and is capable of handling the number of concurrent players you anticipate.

The following steps should be done in order to increase the testing quality.

- Ideally, the bots machines should not be on the internal cluster network. This makes sure the bots connections behave similarly to the way clients connections behave on the production environment and ensures that internal network bandwidth is not being affected by the addition of external bots traffic.
- External latency and packet loss should be enabled on the BaseApp and LoginApps (see the Server Operations Guide for more details). This allows testing real life networking issues while running bot tests.

More information on how to run bots tests can be found at *Stress Testing with Bots*.

Real players should also participate in game testing as part of the normal QA cycle.

2.3. Communicating with the BigWorld Support Team

Maintaining good communication with the BigWorld Support Team is crucial for the successful release of your game. Early reporting of issues will allow us to help solve deployment problems. Communicating your expected beta and release dates will allow us to prepare in advance for the extra effort required to help in releasing your game.

Chapter 3. Cluster Hardware

A typical BigWorld network cluster is built from multiple multi-core machines and a high performance network connecting them. Correct planning of these components will allow for an easy and cost effective implementation. BigWorld recommends installing a test environment using similar but smaller scale hardware as the planned production environment and basing the production environment configuration on the test results acquired from this environment.

3.1. Choosing the Number of BigWorld Server Instances

When releasing your game, you may want to consider running more than one BigWorld server instance.

If you are releasing your game in multiple geographic locations, you may want to host your game in multiple locations so that your servers are closer to the players. Each data centre will need to run at least one BigWorld server.

Even inside a single data centre, there may be reasons to run multiple servers. If your game has multiple shards, you can choose to develop and run your game so that all shards run within a single server instance or you may choose to run a single server instance per shard.

Advantages of running a single server instance include:

1. Machine resources are balanced between all shards.

2. Easier handling of player movement between shards.

3. Ability to send script messages between entities in different shards.

4. Easier to manage as there are fewer databases, fewer accounts to monitor and deploy to etc.

Advantages of running a single server instance per shard include:

1. Not needing to have a single network to run the entire game.

2. Greater isolation of faults. If a single shard gets corrupted, it will not affect other shards.

A solution between these two extremes can also be used having multiple server instances each running multiple shards.

The Server Overview document contains more details.

3.2. The Cluster Hardware

BigWorld recommends using brand name cluster machines with proven stability and performance. Reducing the variations between machines in the cluster can also reduce complex and increase flexibility. The BigWorld engine has not been extensively tested using virtual machines to host the server instance.

Using multiple cores with big L1 and L2 cache sizes is recommended as long as the cores are not overloading the NIC bandwidth. One server component (such as a CellApp or BaseApp) should be run on each core.

3.2.1. General Cluster Machines

Multi-core brand name machines are becoming the market standard and are recommended. We recommend taking into account the specific game requirements and the cost effective market solutions when choosing the cluster machine specifications. In general, our customers are currently using 2-4 core machines with 2-4 GB of memory. One CellApp or BaseApp application should be run on each core. Depending on the usage requirement, each machine should be equipped with one or two 1Gb NICs and with enough disk space to host the server resources, the server binaries and system log files. Appropriate free disk space should be available on the disk where the server is installed. Redundant PSU's should be considered to ensure high reliability.

As of BigWorld 1.9.1, BaseApp processes using the secondary database feature require disk space to write to their SQLite databases. Attention should be paid to ensure that there is enough disk space for these processes.

The machines should have enough memory to avoid swapping as this can lead to poor performance and even processes being shut down. Swap space is still recommended as there is still a chance that these situations can be handled.

For sizing the cluster, please review the section “Sizing the Cluster” on page 10 .

Please note that we strongly recommend that the processor affinity for the server processes be left to the operating system to manage.

3.2.2. Database Machines

The hardware required by your database will be heavily dependant on how much load your database is put under. As with most databases the general rule is to use multiple cores, large memory and high performance reliable hard disks. RAID solutions should be considered for the MySQL hard disk. These should be implemented by a MySQL expert.

Setting up the database machine should be done by an experienced MySQL DBA taking into account the performance measurements taken in the test environment.

3.2.3. Server Tools Machines

The server tools components have greater disk requirements than normal cluster machines. You may wish to have a higher performance hard-disk in this machine to handle the load of logging for all server processes. It can also be a good idea to isolate logs to separate partitions. This ensures that if a single logging process runs away and consumes too much disk space, other processes still have some free disk space.

The general strategy used for long term storage of logs is to use logrotate to perform periodic log switches. Older log files should be archived in order to reduce the space requirements.

3.2.4. Secondary Storage Considerations

It is crucial that the master copy of the server binaries and resources and the MySQL database are stored on fault tolerant hard disks which are also backed up regularly.

BigWorld recommends against using NAS related solutions like NFS for the database as these solutions can cause increased network load and therefore degrade performance.

The BigWorld cluster has yet to be performance tested using SAN solutions but we expect these solutions to better fit the BigWorld cluster deployment.

When deploying the BigWorld cluster on separate machines without the usage of a SAN or a NAS solution, a solution should be used to distribute game resources between the cluster components. Any commonly used solution can be used for this issue. Some examples include:

- rsync
- puppet (http://reductivelabs.com/trac/puppet)
- cfengine (http://www.cfengine.org)

- custom scripts

3.2.5. Sizing the Cluster

BigWorld Technology supports multiple types of games. Some games include many NPC entities and complex server AI calculations while others include more player characters and less NPCs. The number of spaces can also vary between different implementations. This variety means that the cluster requirements will depend on the specific customer game implementation.

The best solution for sizing the cluster is to measure the BigWorld server instance requirements on a test environment and then to use these results to predict the required cluster hardware. BigWorld Technology allows adding machines to a cluster as well as changing the role of each machine. This allows easy adjustment of the cluster during both the testing and the deployment stages. Please note that extra resources should be allocated to any server to handle load spikes and unexpected events. The chapter *Cluster Deployment Examples* on page 17 below contains examples based on real cluster deployments.

3.2.5.1. Cluster Machines

In order to establish the hardware specification required for a game, tests should be run in a representative test environment. We recommend creating a representative cluster which will include the same ratio of CellApps, BaseApps and other processes as planned for the production environment. A WoW style game should probably have a test environment for at least one complete shard while any style game should aim to have a test environment sized at least 10% of the planned production environment.

Scaling from the test environment to the production environment should be done in small steps to detect any bottlenecks or unexpected issues. This includes identifying changes that may need to be made in the game's script and other resources. In some cases, it may even require changes to the game design so early detect of these issues is important.

The testing stage should try to estimate the expected production environment while also monitoring less scalable components (like the CellAppMgr, BaseAppMgr and database components) to detect potential bottlenecks.

3.2.5.2. Cluster Network

The external-facing cluster machines would typically be connected directly to the Internet without the usage of any hardware based firewalls. The estimated required bandwidth per machine should be calculated based on the results of the test environment and extra bandwidth should be allocated to prevent problems due to network spikes. A rough estimate can be achieved by multiplying the requested downstream bandwidth per client by the expected number of clients. Extra bandwidth should be allocated for resending dropped packets as well as handling temporary spikes in the amount of data being sent. Outgoing bandwidth is generally higher than incoming.

In order to establish the network requirements, it is recommended to run the same network cluster as above and monitor network bandwidth and loss. Additionally, the network switch load should be monitored using switch monitoring utilities or other network interface monitoring utilities to measure the internal cluster bandwidth requirements and internet bandwidth requirements.

BigWorld Technology has multiple mechanisms to overcome packet loss issues but it is recommended that the underlying network error rate is within acceptable parameters (for example, less than .01% in unsaturated network conditions).

3.2.5.3. The Bots Process

The bots process should be used early and regularly to test with large numbers of simulated players. Time should be spent to create bot scripts that will simulate player behaviour. Server load and bandwidth usage should grow near linearly with the number of bots added. Attention should be paid, if this growth does not appear to be linear as this can lead to scaling issues in larger production environments.

3.2.5.4. Staged Release

It can be difficult to estimate the expected player base and expected concurrent player count. It may make sense to release your game in a stage manner to help predict the peak usage and cluster requirements as well as having an opportunity to identify any problems earlier with a smaller player base.

3.3. Cluster Configuration

3.3.1. Linux Distributions

For long term deployment environments, BigWorld supports and recommends the usage of one of the following 32 bit Linux distributions:

- Debian (Stable) http://www.debian.org/
- RedHat Enterprise Linux http://www.redhat.com/rhel

3.3.2. Smoothing load during startup

While some effort should be put into tweaking server configuration for operational stability, it is also important to consider the load of your system during server startup. Spiky load during server startup may reduce the speed that your cluster will startup and, in the worst case, potentially prevent server startup completely due to overloaded processes.

In order to assist spreading load between server processes and increasing space load time, the following options should be configured in your game's `bw.xml` file.

- `<desiredBaseApps>`
- `<desiredCellApps>`
- `<cellAppMgr/maxLoadingCells>`
- `<cellAppMgr/minLoadingArea>`

While these options will assist in smoothing out your server startup, it is important to take into consideration that the CellAppMgr configuration options apply across the entire lifetime of the server, not just during startup. For this reason it may be useful for your game script to implement a loading phase and an active game phase. During the transition between these two phases the configuration options such as `<cellAppMgr/maxLoadingCells>` can be modified via a Watcher set request. This also applies to any other load balancing options or other configuration options you may choose to tweak for game startup. See the Server Operations Guide for more details on these options.

3.3.3. MySQL Configuration

MySQL is quite well configured by default, however there are a few options that should be considered for tweaking to ensure optimal performance within your own production environment.

`innodb_buffer_pool_size`	MySQL documentation suggests that this system variable can be set to 80% of the physical memory size. However, you must also take into account that the total memory available to a 32-bit process under Linux is 3GB. And within that 3GB, the heap can only occupy 1.8GB. So even if the machine has 4GB of physical memory, `innodb_buffer_pool_size` should not be set to more than 1.5GB. Otherwise, "out of memory" errors may occur when MySQL server is under heavy load.

3.3.4. Network Configuration

The cluster network should be carefully configured to provide both security and performance.

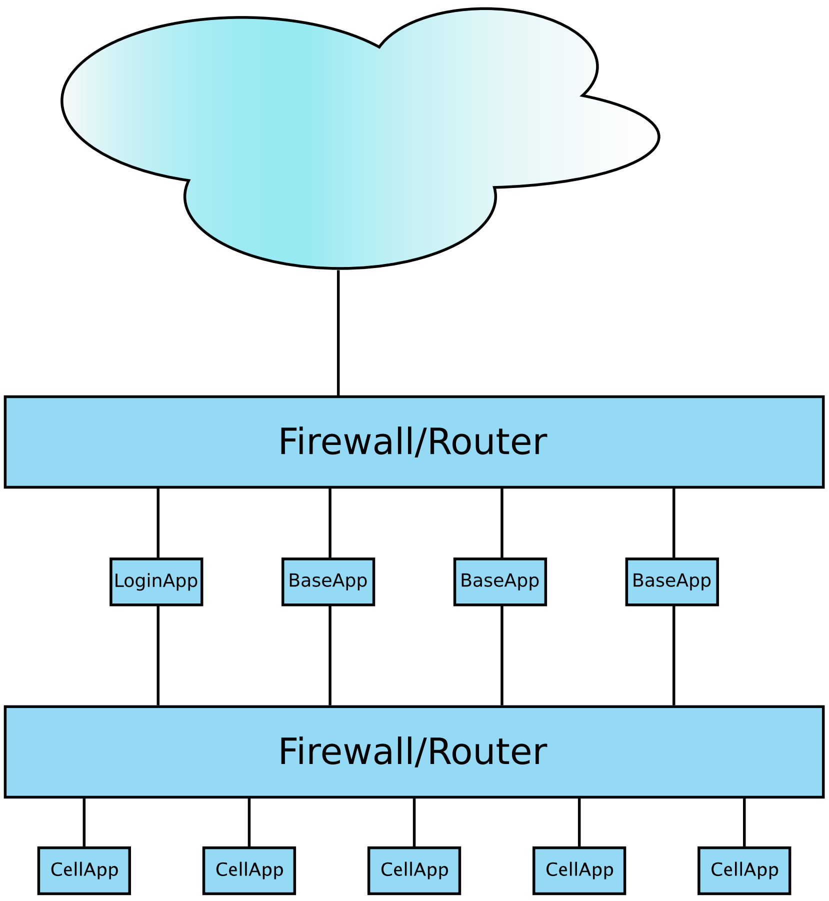
3.3.4.1. Installing Multiple LoginApps

BigWorld recommends using DNS round robin in order to achieve load balancing when deploying multiple LoginApps. There are many good references available on the Internet explaining how to install and use this DNS feature.

3.3.4.2. Security Considerations

3.3.4.2.1. External Network Security

A typical BigWorld cluster includes multiple external facing machines as displayed in the image below. These machines include both BaseApp and LoginApp machines which are used for the client login and for communication with the other server components. All external facing machines should be carefully configured to prevent hackers from getting access to these machines. Additionally, BaseApp and CellApp Python code should be carefully reviewed to prevent any potential exploits. For further information please review Server Overview.



BigWorld Server Instance components

3.3.4.2.2. Communication Encryption

BigWorld Technology ships with a default LoginApp RSA key pair to enable encrypted login communication, however all BigWorld customers receive the same pre-generated key pair. In order to secure the client connections for a BigWorld game, a new key pair should be generated and the public key distributed with the game clients. The key size directly impacts the performance of LoginApp. For instructions please see Server Programming Guide.

3.3.4.3. Default broadcast network routes and External Interfaces

3.3.4.3.1. Configuring internal interfaces

In order to ensure the correct operation of your cluster, it is important to correctly configure network routing on every host. From BigWorld's perspective, this means making sure that machines with multiple network interfaces (such as BaseApp and LoginApp machines) have the correct default network route established for broadcast network messages. BigWorld servers use broadcast messages as a mechanism to establish internal network interfaces as well as finding other servers in the cluster. Please review the Server Installation Guide for more details.

3.3.4.3.2. Configuring external interfaces

While default broadcast routes are used to identify internal network interfaces it is necessary to configure BigWorld servers to know which interface to use for external network communication. In production environments, the Internet facing network will generally have a separate network assignment to the internal network.

To configure your cluster to use the correct network interface use the `bw.xml` option `<externalInterface>`[1]The recommended method for specifying external interfaces is with an IP address / netmask combination to avoid having individual `bw.xml` files per host. For example using the previous example host configuration we could add an entry as follows:

```
<root>
    <personality> FantasyDemo </personality>

    <parentFile> server/production_defaults.xml </parentFile>

    <externalInterface> 192.168.1.0/24 </externalInterface>
</root>
```

Although not strictly necessary, it is recommended to standardise the network interface assignment to assist with system administration. For example, all internal network interfaces could be configured as `eth0` and all external interfaces should be configured as `eth1`.

3.3.5. BWMachined Version Consistency

While BigWorld strives to make bwmachined versions interact with each other as seamlessly as possible, it is recommended to make sure that all the installed versions of bwmachined within your BigWorld network cluster have the same version number. The easiest way to check this is using the WebConsole | All Machines option. WebConsole will display differently versioned bwmachined installation in red and will show a warning at the bottom of the screen if multiple bwmachined versions are installed within the same cluster.

Newer bwmachined daemons are developed to support running older instances of the server, so updating to a newer version of bwmachined will still allow running older servers.

3.4. Client Hardware Recommendations

The requirements for running the client are heavily dependant on the visual complexity of your game and the number of effects (such as particles) that are used for different quality levels. The following recommendations are based upon the FantasyDemo game environment and client as shipped with the default BigWorld

[1]For more details see Server Operations Guide *Server Configuration with* `bw.xml` "General Configuration Options".

packages. We recommend running client performance and stability tests on different platforms to ensure compatibility and performance of the client on these platforms.

The other major factor that can determine client hardware requirements is the kind of terrain used within your game. For more information on the different terrain types available please refer to Client Programming Guide's chapter *Terrain*.

3.4.1. Advanced Terrain Recommendations

Recommended	Minimum
GeForce 7600 or ATI Radeon x1600	GeForce 6600 128 MB or ATI Radeon 9600 128 MB
2 GHz CPU	2 GHz CPU
1 GB RAM (2GB if running Vista)	512 MB RAM
Windows XP Home 32bit or Vista Home 32bit	Windows XP Home 32bit

3.4.2. Classic Terrain Recommendations

Recommended	Minimum
GeForce4 Ti or ATI Radeon 9500	GeForce4 MX or ATI Radeon 7 Series
2 GHz CPU	2 GHz CPU
1 GB RAM	512 MB RAM
Windows XP Home/Pro 32bit or Vista Home 32bit	Windows XP Home 32bit

Chapter 4. Monitoring and Maintaining the Cluster

After deploying the BigWorld cluster, an operations team should be in charge of maintaining and updating it as required.

4.1. Day To Day Monitoring

The operations team should monitor the cluster usage on a daily basis using StatGrapher to detect unexpected issues. Disk usage should also be monitored as running out of disk is a common cause of multiple server issues particularly for tools machines.

The operations team should pay attention to any server crashes and report those to the BigWorld support team making sure all relevant information is collected and sent to the team. See the Server Operations Guide for more information about the procedure for reporting a crash.

Chapter 5. Communicating with the BigWorld Support Team

Maintaining good communication with the BigWorld Support Team is crucial for the successful release of your game. Early reporting of issues will allow us to help solve deployment problems. Communicating your expected beta and release dates will allow us to prepare in advance for the extra effort required to help in releasing your game.

Chapter 6. Cluster Deployment Examples

Provided below are estimates on the cluster hardware required for different deployments for an RPG-style game. These estimates are based on historical cluster deployments but will change based on each specific game characteristics.

6.1. Cluster Machines Distribution

The following calculations are used to estimate the cluster size for a specific deployment

- **Number of Cores for CellApps**

  The ratio between concurrent connections and CellApps Cores ranges between 100:1 and 300:1.

- **Number of Cores for BaseApps**

  The ratio between CellApps and BaseApps Cores ranges between 6:1 (six times more CellApps) and 1:1 (same number of BaseApps to CellApps).

- **Number of Cores for BaseAppMgr, CellAppMgr and DBMgr**

  One core should usually be used for each of the processes BaseAppMgr, CellAppMgr and DBMgr.

- **Number of Cores for LoginApp**

  One core should be used per LoginApp. At least one redundant LoginApp is recommended.

- **Number of Cores for MySQL**

  At least two cores should be used for the MySQL database.

Assuming a 3:1 ratio between CellApps and BaseApps and taking 8 cores extra for other processes including the MySQL server, The number of cores required to support 5000 concurrent users ranges between 30 and 74. If using 4 cores machines this ranges between 8 and 19 machines.

An example Blade setup would be as follows:

BladeCenter LS20 885051U

- Processor: Low Power AMD Opteron Processor Model 246 (Standard)
- Memory: 4 GB PC3200 ECC DDR RDIMM (2 x 2 GB Kit) System Memory
- IBM eServer BladeCenter (TM) Gigabit Ethernet Expansion Card
- CSI Hard disk drive 1 : 73GB Non Hot-Swap 2.5" 10K RPM Ultra320 SCSI HDD

BladeCenter 86773XU

- Optical device: IBM 8X Max DVD-ROM Ultrabay Slim Drive (Standard)
- Diskette drive: IBM 1.44MB 3.5-inch Diskette Drive (Standard)
- Power supply modules 1 and 2: BladeCenter 2000W Power Supplies one and two (Standard)
- Management modules: BladeCenter KVM / Management Module (Standard)
- Switch module bay 1: Nortel Networks Layer 2/3 Copper GbE Switch Module for IBM eServer BladeCenter
- Switch module bay 2: Nortel Networks Layer 2/3 Copper GbE Switch Module for IBM eServer BladeCenter

A setup with 10 LS20 blades in a 86773XU BladeCenter would cost approx $60k USD.