

Distributed Cell Architecture

Jonathan Morales

Shadowrun: Awakened, 2007

www.awakenedmmo.org

1. INTRODUCTION

In the development of a massively multiplayer online game it becomes important to pay critical attention to the development of the server. While monolithic single server architectures provide simplicity and speed in some scenarios, they fail spectacularly in scalability, robustness and accessibility. Distributed architectures provide scalability, robust design and a component-based system where individual component failures will not bring down the entire system. Distributed Cell Architecture (DCA) aims to provide a custom solution to meet the needs of the Shadowrun: Awakened online universe.

2. COMPONENTS

Distributed Cell Architecture is a combination of diverse systems which work in synergy to power a vibrant virtual world and deliver robust content to the end client in a timely manner. Systems within the DCA architecture are designed to be modular and, with the exception of the Master Cell, are largely capable of autonomous functionality.

The first and primary component of DCA architecture is the Master Cell. The Master Cell functions as the brain of DCA. It is the client's first contact when connecting to the simulation and acts as the manager of the entire distributed architecture. Additionally, the Master Cell handles load balancing across the DCA as well as forwarding, execution and verification of administrative commands.

The second and vital component is the Zone Cell or simply Cell. The Cell handles the running of the simulation and its distribution to the client. The

Cell handles all updates to the simulation as well as rulesets and game mechanics. To enforce isolation, Cells retain internal logging and database access faculties through standard interfaces. Cells can communicate with each other in an imperative one-way manner via a standard message protocol. While Cells rely on the Master Cell for direction and management, they are capable of sustaining themselves indefinitely and are thus a reasonably unbounded non-deterministic system.

The third component of DCA is the Chat Server. The Chat Server is employed as a separate unit from Zone Cells separating clients to avoid network congestion and resource contention. Additionally, being a separate component allows future customers external to the simulation to communicate with 'in-simulation' clients via a separate chat application. The Chat Server is developed to maintain a small footprint while keeping traffic flowing as freely as possible. Additional mechanisms are employed to allow caching and artificial slowdown of message delivery in order to ease the burden on simulation should Zone Cells become taxed. The Chat Server maintains a user list which is linked to player coordinates as well as group membership.

The fourth central component of DCA is the relational database. The database handles final recording of all transactions. Only transactions within the database are considered complete and valid. The database is responsible for both live data and backup data as well as logs submitted by Cells on a nightly basis. The database provides consistency and storage of data across the larger simulation. Additionally, the database powers web-based administrative and client account tools

while providing statistical information for character stats and simulation-wide growth and economics.

The fifth and final component of DCA is the Local Utility Service (LUS). The LUS is installed on all servers running Zone Cells. The LUS handles waking up and rebooting of Cells as well as writing of Cell Data to disk. This includes Log Files queued via the Zone Cell log interface.

3. MASTER CELL

The Master Cell functions as the brain of DCA. It is the client's first contact when connecting to the simulation and acts as the manager of the entire distributed architecture. The Master Cell encompasses login, character handling, load balancing and directing of clients and groups.

3.1 Login

The main goals of revision three of DCA (formerly known as SCDA) were to simplify the architecture and reduce potentially redundant components. Modules were surveyed for performance, isolation and purpose. Components found to not meet these requirements were reviewed for possible inclusion in parent or related systems. The Login Server was the first module to be moved to a parent module. The Login Server will generally handle small burst traffic with very small payloads. Having a separate system to manage this low traffic was thought to be wasteful as well as redundant as the Login Server would have to pass information to the then Session Server (Master Cell) in order to move the client into the simulation.

To reduce complexity, the Login Server was merged into the Session Server and the Master Cell was created. The Master Cell serves as the central entryway for clients entering the simulation cluster. Upon initial connection, the client provides both username and password credentials. These are independent of character names, of which the client may have many. These credentials are then matched against those stored in the client database. If valid, a token is placed in the database as proof of client authentication. A copy of this token is returned to the client and

serves as an encryption key should secure transactions be needed during the current session.

3.2 Character Handling

After a successful login, the client is directed to create or select a character. If the client has previously created a character, the Master Cell retrieves the client's selection from the database and prepares to transfer the user into the simulation. If a character has yet to be created, the server will hold the client's connection open up to thirty minutes while character visual customization is performed via client software. When finished, the server receives the resulting combinations. The server then receives character statistical information and performs verification against a pre-determined character ruleset. If the verification fails, the client is potentially hacked or contains a developer bug. The server will destroy the temporarily stored character and request that the client begin the process over. If character statistical information is in fact valid, the server will save the character to the character database and prepare to transfer the client into the simulation.

3.3 Load Balancing

Post character selection, the client will be directed to the Zone Cell where their gameplay session previously ended or to a starting Zone Cell if they created a new character. At this point, if configured so, the Master Cell may opt to perform load balancing. The Master Cell retains a list of all clients who are currently logged in as well as their current locations. The heuristic for determining whether to spawn a new server is player count followed by ping response time if the Zone Cell is near player max. If a Zone Cell is overtaxed, the Master Cell will check its server listing and determine, by means of total player count, if any servers are currently experiencing limited usage. If a suitable server is found, a request is made to the server's host LUS service to start up a new Zone Cell with the specified map. In this way, the simulation cluster achieves dynamic load balancing.

3.4 Directing Clients

Once a suitable Zone Cell has been selected or created for the client, the transfer process begins. The Master Cell updates the database with the client's currently selected character. The Master Cell then notifies the Zone Cell receiving the client of an impending transfer. Once confirmed, the Master Cell then notifies the client of its new IP and port to connect to. The Master Cell then disconnects the client, ending the transaction. After one minute, the Master Cell will check the client's entry in the database and will delete their authentication token should their entry not be updated with a receipt token from the Zone Cell.

3.5 Directing Groups

If a load balancing solution is implemented, special care must be taken to maintain group integrity when moving clients amongst Zone Cells. The Master Cell retains a list of all clients and their resident zones as well as group associations. If a client requests transfer to another zone, their group status is checked. The Master Cell will attempt to transfer the client into the same instance as their group. A Zone Cell reserves the right to refuse a connection to maintain local stability. In this scenario, the client will be forced to transfer into another instance of the Zone until such time when the primary Zone Cell feels it has cooled down.

3.6 Cell Map

In order to direct traffic between cells and clients, the Master Cell retains a table known as the Cell Map. The Cell Map contains the relationships between Cells and Server IP addresses. It also contains a count of the number of clients per cell for load balancing purposes.

3.7 Client Map

To complete fulfillment of command and control throughout the cluster, the Master Cell retains a table matching Cells to Clients and Clients to Groups. In this manner, the Master Cell can track the location of every player in a general manner. This is useful for cluster-wide client administration purposes.

3.8 Administration

Perhaps the most useful function of the Master Cell is administration. The Master Cell is the only entity with global knowledge of the distributed cluster. With this knowledge, the Master Cell can handle server, cell, client and player administration. Administrative tools or commands send messages to the Master Cell, which then forwards these messages to the appropriate server, cell or client on the cluster.

3.9 Cell Messaging Protocol

Cells have the ability to pass messages in four main formats.

General General messages are Cell to Client Messages. These messages can be grouped under Non-Terminal, GameState and Transact. Non-Terminal are the lowest priority messages and receive lowest status on resend queues and may also be dropped completely. GameState messages represent anything from player position to combat information. GameState information must generally be received but will not result in data corruption or can be lost without substantial impact. GameState messages are standard priority. Transact messages represent things like player-to-player transactions (trade), in-game banking and other situations where data must be received or the transaction is deemed void. Transact messages always have highest priority and receipt is always verified.

Priority Alert Priority Alerts are system-wide messages which must be delivered to all clients. Since Priority Alerts are essentially broadcast messages, a staggered release schedule is assigned. Priority Alert messages can only be sent by the Master Cell or Session Server. Examples of Priority Alert messages are notice of system shutdown.

Global Action Global Action messages are part of inter-cell communications. Global Action messages always originate from the Master Cell but can be responded to by any Cell. Global Action messages can be assigned a priority of Low, Medium or High and will be responded to accordingly. Example Usage: Admin wants to ban "JoeDude001". Admin tool sends message to

Master Cell saying “find JoeDude001. Ban him and send Receipt to Admin”. The Master Cell broadcasts a Global Action message to all cells. The Cells who do not have JoeDude001 do not respond. The Cell who does have JoeDude001 responds and performs the action. A receipt is sent back to the Admin’s IP. Global Action messages are always considered critical and require receive receipts.

Core Core messages are part of inter-cell communications. Core messages are passed between a Cell and its neighbor via dedicated TCP connections.

4. ZONE CELL

Zone Cells represent a discrete map segmentation in the game world. Zone Cells break up what would otherwise be a very large world map into smaller manageable pieces. Zone Cells perform all logic and game mechanics within their simulation. The various Zone Cell simulations as a whole are taken together to form the SRA universe. The interior of a Zone Cell can be referred to as a cell stack. It is a collection of 4 different systems which work in unison to serve and manage the simulation.

4.1 Cell Logging

Zone Cells have the ability to log local transactions to a universal logging format. The Zone Cell’s internal logging interface receives log messages from any layer in the cell stack ranging from Simulation to Network Communications. The messages are placed in a queue to be handled by the local server’s LUS, which manages log messages from all Zone Cells on the local server.

4.2 Database Interface

All Cells use a standard database interface to call remote procedures or execute tailored queries. The database interface is designed to provide SELECT, UPDATE and DELETE services to the Cell.

4.3 Simulation

The simulation is the game world. The simulation represents the terrain, players, non-player characters and all other game objects.

Additionally, it also represents the rules and game mechanics which drive the SRA universe.

AI and NPCs There are two general categories of AI control, team-based and single. Team-based AI NPCs are led by a command NPC and implement team-based tactics. Single AI units attempt basic maneuvers on occasion such as flanking or suppression. AI can be further divided into combat and common. Common AI-driven objects tend to flee or seek other-than-combat means to resolve conflict. Combat AI is focused on the destruction of the target enemy. NPCs are capable of switching between common and combat AI. Using a combination of stances and objectives, the AI can be made to accomplish diverse tasks. AI comes in five general stances:

Aggressive – This AI will attack its target on sight.

Hostile – This AI will attack its target on provocation.

Neutral – If provoked, this AI may flee or attack if armed (the most common stance).

Friendly – This AI may assist the player in combat. May attack if provoked enough.

Ally – This AI will not attack unless provoked several times. This AI will always assist the player.

AI states come with one or more objectives:

Follow – AI attempts to follow someone.

Hold – AI attempts to stay in an area. It will defend itself if stanced correctly.

Goto – AI attempts to go to the destination.

Kill – AI attempts to kill a specific person.

Protect – AI attempts to protect a target.

Combat In the Shadowrun: Awakened world, combat can take many forms. Some of the most prevalent are melee, ranged and magical. From a server perspective, once combat has engaged, common activities such as trade are prohibited. Combat is implemented in a pseudo turn-based format. Each round, an action queued from the client is processed and resolved. Only one action may take place in a round unless otherwise flagged. The server performs rolling of dice to

determine hits and application of modifiers as appropriate. Because combat is fast-paced, results are saved to the database after each encounter. Should the client disconnect or otherwise leave the simulation, the client's current data is written to the database. This is standard practice and occurs anytime a player leaves the simulation or transfers zones. Lastly, line of sight checks for combat are performed on the server with results being forwarded to all players in range.

Collision Comparing the player's bounding box against local map geometry performs collision within the Simulation. NPCs travel along predetermined lines, which take care not to intersect with map geometry. AI may perform collision detection on the fly when determining destinations. On a collision with a motionless object tagged as permanent (e.g., buildings), the player comes to rest against the object's collision box. On collision with a moving object or a target with limited mass, the target or the player may be pushed in the appropriate direction as per the physics engine. Movement-based collisions will never knock or cause damage to the player.

Economics and Trade Trade within the simulation is handled via a guaranteed message. A trade is not finalized until both parties have agreed to the transaction. NPCs automatically agree or disagree depending on standing with the player. All trades are final. Players may not trade across Cell boundaries. Players may only trade with an individual in their immediate vicinity; this is enforced via server checks.

Items All items within the simulation originate from the Zone Map used to start the Cell. The simulation is capable of creating items on demand when requested by a client. Once acquired, item ownership is flagged as belonging to the client.

Team System Players who decide to group form a team. A team may contain no more than eight characters. Grouped players are flagged and their status updated in the Master Cell. Grouping consists of a two-stage request and accept process. Group status is not saved and is terminated when the client logs off. Players who are grouped

receive special consideration when encountering AI.

Players in a team must be constantly aware of each other; as a result, grouped members receive updates on member health and status conditions. This allows the client to generate health information for teammates.

Organization System Organizations are groupings of players and NPCs on a larger scale. Corporations and organizations are pre-existing and can be joined by players with appropriate faction ranking.

Membership into certain organizations can automatically make one an enemy of another. Organizational affiliation is a permanent status which is tracked in the database and retained when logging off.

Physics Physics is employed via Crystal Space or other physics modules. All objects in the simulation institute physics to varying degrees. The Zone Cell avoids performing pure physics calculations whenever possible.

Object Inspect A player may choose to inspect objects in the game world and view textual descriptions of each. These are stored locally. If a player selects another player and requests information, the server will generate limited information based on that target player's privacy settings and/or privacy equipment/gear. These updates are then sent to the client.

Missions Missions are of special interest as they are central to the gameplay of the simulation universe. Missions received and completed are stored in the database and are persistent across sessions. Missions come with lenient timers; upon login, mission timers are checked and expired missions are removed as failed.

Missions are employed within the zone with free access to all players. Missions can span any number of Zone Cells within the simulation. To accommodate multiple players, mission objects are singular but experience a fast respawn rate. Mission objectives can be randomly created within a set area if more than one player is in the same area performing the same mission.

World Save All character data is written to disk every 30 minutes. At midnight, characters are written to the database utilizing a staggered schedule. Rather than delete the characters on disk afterwards, these copies are maintained for possible retrieval. This ensures that a loss of a physical hard disk from any cell can be mitigated using the database. Likewise, should the database become corrupt, characters can be retrieved in their most recent state from the disk on each Cell. The storage space for these characters is minimal and can never exceed the total number of clients on the cluster. The maximum amount of information loss possible is one day. This is an acceptable loss.

Map and Mini Map In a change from the second revision of DCA, clients will now handle rendering and populating of maps and mini-maps.

Daylight Cycles, Weather and Sound Night and day cycles are simulated on the server every 4 hours. Current time in the simulation is passed to the client on connection and the client is responsible for simulation of time passed. The time is re-synchronized every time a new zone is entered. Weather state is simulated on the server with a percentage chance of each state. Weather states are calculated every 30 minutes. A single message is passed to the user to modify weather states. Sound is simulated purely client-side.

Death In the dangerous and complex world of Shadowrun: Awakened, death is a common and devastating end to gameplay. There are multiple possible outcomes to a death scenario depending on gameplay options chosen and pure random chance. If a player is killed but escapes death via one of these random options, the character is flagged as unconscious and gameplay resumes at the appropriate time. If in fact a character has no escape from death and random chance has not fallen on their side, the character is flagged as being dead. Time of death is noted in the database. Characters flagged as being dead remain in the database and may appear in the character selection screen until manually deleted by the client. Clients may choose to challenge causes of death if they believe it was due to

circumstances beyond the simulation (disconnect).

Client Object The client is represented in a Zone Cell's memory as a client object. The client object contains all data on the current character to include stats, inventory, nuyen, standings with factions and mission list.

When traveling to a different zone the client object is destroyed and a new client object is created upon successful connection to the new zone. This object is created from the database to ensure data integrity.

Inventory Both NPC and Client inventories are maintained within the simulation on the server. At logoff or zone transfer, the player's inventory is written to the database. A client may have an unlimited amount of items, game mechanics permitting.

4.4 Cell Transitioning

Because the SRA universe is overlaid across a variety of Zone Cells, swapping Cells is a natural and common occurrence for the client. When a client reaches the outer limits of a Zone Cell, the host cell notifies the receiving cell of impending transfer. The client's authentication token is passed to the neighboring Cell. The client is then disconnected and handed off to the neighboring Cell. The host Zone Cell notifies the Master Cell of a client transfer. This allows the Master Cell to invalidate the client's token should they take too long to connect to the next cell. Upon successful connection, the receiving Zone Cell updates the database to verify receipt. The Zone Cell also notifies the Master Cell of client receipt. In this manner, the client's connection and authentication token are never left open should they randomly drop. The constant token checks also ensure that clients can only enter the cluster via the Master Cell.

4.5 Cell Security

Cell security is the last line of server defense. Cells work on what can loosely be termed a referral system. All clients received have been received via a connection from another trusted cell. Trust is established via the Cell Map which

defines Cell Relationships, addresses and encrypted communications tokens. Cells do not accept clients which have not been referred by another cell. The Master Cell acts to bring the client into the referral network.

4.6 Fallback Action

By default Cells will close access to a zone and notify the Session Server should a specific Cell drop from the simulation. Cells ignore all requests originating from an external network. Communications between cells are based on an encrypted token located on the Cell Map and generated by the Master Cell.

4.7 Simulation Communications

The server simulates the game in discrete time steps called ticks. Cell Servers in DCA simulate the game world at 10 ticks per second. This means that every 100 milliseconds the server processes all user input, takes a snapshot of the game world and updates players as appropriate. Client input is sampled 10 times a second as well. This is fast enough to calculate MMO transactions since client input is not typically received as fast.

In contrast, popular first-person shooters sample input 33 to 100 times a second with updates occurring every 10 to 30 milliseconds.

Every 100 milliseconds the server should loop through all of the packets it has received (user input from the clients), verify them, apply them to the game state and send out the new game state to all affected clients.

In general, actions do not occur until a response has been given by the server updating the client's local simulation. The exception to this rule is movement. The client is free to move wherever they please without server authorization. The client sends movement updates to the server every tick. The server sends updates to the client on where it has tracked the player. If the 'real' simulation on the server deviates too far from the 'client simulation' the client is corrected. Ultimately, the only positional coordinate which matters is the one on the server. A hacked client which declares itself several yards away in a

millisecond is declined by the server and its actions logged.

For the client side, a small degree of interpolation and extrapolation may need to be done in the case of fast moving objects to avoid 'jitter' effects. One such method is Dead Reckoning. Designed by the defense department for use in the Distributed Interactive Simulation protocol, Dead Reckoning is a system in which everyone participating in a game simulates all objects. These objects, typically vehicles, are simulated at a coarse level of fidelity. The basic idea behind Dead Reckoning is that all parties agree on a set of algorithms which are used to extrapolate the behavior of objects in the game. Additionally, a set of constraints is agreed upon which dictate how far reality can deviate from this extrapolation. If an object deviates too far, a correction is issued. Dead Reckoning is a form of client-side prediction.

5. CHAT SERVER

The Chat Server is responsible for passing messages between connected clients and is employed as a separate unit from Zone Cells. This serves to avoid network congestion and resource contention. Additionally, being a separate component allows future customers external to the simulation to communicate with 'in-simulation' clients via a separate chat application. The Chat Server implements a type of flow control to minimize stress on the server.

5.1 Chat Configuration

The Chat Server maintains a list of all clients connected to the simulation. Clients are responsible for manual update of their coordinates and current zone. This information is used to forward messages to the correct users. On startup, clients forward lists of friends (based on character), global friends (based on account), ignore and blocked individuals. The Chat Server uses these to filter packets before routing them to their destination, avoiding unnecessary network traffic. Clients may create chat groups. When a chat group is created, a list of members is created on the server, which is maintained until all

members have logged off. This list serves to filter traffic sent to the chat group members.

5.2 Chat Filtering

Changed from SCDA, chat bad-word filtering is performed on the client.

5.3 Flow Control

If the Chat Server receives instruction to or if it determines (based on a set heuristic) that the network is becoming congested, the Chat Server may choose to implement flow control. Flow control is simply a forced delayed delivery of packets to the client.

6. RELATIONAL DATABASE

The database handles final recording of all transactions. Only transactions within the database are considered complete and valid. The database is responsible for both live data and backup data as well as logs submitted by Cells on a nightly basis. The database provides consistency and storage of data across the larger simulation.

6.1 Database Layout

The SRA database is responsible for storing the following information:

- Client Data
- Character Data
- Character Statistics
- Character Inventory
- Visual Cue IDs (color, textures, model, hair)
- NPC Data/Statistics/Inventory
- Character Lifetime statistics
- Missions current and completed
- Backup Cell Logs

The SRA database is a relational database backed up to DVD once a month or once a week.

7. LOCAL UTILITY SERVICE

The LUS is installed on all servers running Zone Cells. The LUS handles waking up and rebooting of Cells as well as writing of Cell Data to disk. This includes Log Files queued via the Zone Cell log interface.

7.1 Cell Control

The LUS is a small footprint service that has the capability of performing a small selection of Cell maintenance and administrative functions. Chief among those are startup of a remote cell and forced reboot of a remote cell. These commands are issued from the Master Cell and arrive encrypted.

7.2 Logging Components

In addition to logging, the LUS maintains a data buffer which, when filled, will trigger a write phase in which the LUS will log all current messages to disk. The LUS implements a unified logging format with common fields allowing easy archival and storage. The LUS is also responsible for transferring log files from disk to the database for long-term archival.

8. TOOLS

An important part of management of the Distributed Cell Cluster are specially developed tools. Tools can be broken into five categories: Server Management, Cell Management, Player Management and Account Management.

8.1 Server Management

Server management consists of tools which directly affect the server which is hosting Cells and other systems. Server management is accomplished via the Local Utility Service to accomplish features such as restarting of cells and requests for resource information.

8.2 Cell Management

Cell management is done via the Master Cell. The Master Cell can request data about any object in the simulation. Cell Management is the foundation of player management.

8.3 Player Management

Player Management involves the direct manipulation of the player and assets related to them. Because of the distributed nature of the cluster, player management tools interface with the Master Cell to retrieve information on player location before issuing commands to the resident Cell for action.

8.4 Account Management

Account management consists of web-based tools to administer client accounts.

8.5 Site Tools

Site tools allow for the creation of interactive websites featuring character stats, lifetime stats, game world stats and even snapshots of the client's characters.

8.6 GM Tools

GM Tools allow simulation administrators to perform routine duties such as moving stuck characters or creation and deletion of items in game. GM tools should be used by trusted game administrators. GM Tools log the creation and distribution of admin created items to combat admin assisted cheating.

9. PATCH SERVER

The Patch Server is labeled as an optional system due to the fact that there are somewhat cheaper methods for patch distribution. The Patch Server is not concerned with authentication and can even be developed later as a separate entity. The Patch Server, if implemented, will accept connections up to its peak bandwidth. It will receive a client connection and perform a checksum of the client's critical files (everything except graphics and sound). If the client is missing files or they are the wrong version, the Patch Server will send those files to the client via an FTP stream. Upon successful patching, the Patch Server will tear down the connection. The client at this point should either restart or return to login.

10. STARTUP PROCEDURES

DCA utilizes a top-down approach to startup of the distributed network. The Master Cell (or Top) begins the startup process. On startup, the Master Cell sends a small message to the Local Utility Service on all registered servers saying "startup all cells".

Upon starting up, the cells would send a message to the Master Cell saying "here I am, register me". The Master Cell would register the response and

distribute an encrypted communications token. Cells will then start their simulations. Once started, the Cells message back saying "I'm running". The Master Cell will then update its master table (or Cell Map) and begin accepting and directing client logins.

11. A SAMPLE CASE

The following is a sample case designed to show components in action at a higher but practical level. It is assumed that the player has successfully logged in at this point.

The Master Cell, having read the player's token and verified it against the database, determines which Zone Cell the player must head to based on the Cell Map. The Session Server, in our example, determines that this player left off in Cell A. It matches Cell A to IP .25 and sends a message to Cell A notifying it of client transfer. The client connects to Cell A and begins playing after loading resources, etc. Each Cell, being a zone in the game world, is self-sustainable. The best way to think of it would be a 'level'. The same engine and concepts are used but a different 'map' or 'level' powers every Zone Cell. Whichever way it is visualized best, the Cell is a self-sustaining entity.

Let's now assume that the player heads west and comes to the zone boundary. The server checks its Cell Map and determines that the connecting cell is Cell B. It sees that Cell B resides at .25 or localhost (same machine). The player is disconnected from Cell A and Cell B proceeds to regenerate the player's object from the database. The player connects to Cell B. After loading the zone, play continues. The new player location is stored in the database since the load was successful. Now let's say the player continues west still further! The server checks its Cell Map again and determines that the adjacent Cell is Cell C located on machine .28. Cell B on .25 then contacts Cell C on .28 and notifies it of a client transfer. The client disconnects from .25 and connects to .28 Cell C. Cell C, having recreated the client's information, accepts the new client connection. The player loads Cell C data and play begins again. His new location is logged in the database. To the player, he's simply zoned

multiple times while moving west. On the server he's changed his physical connection and now belongs to server .28. The .25 machine represents what our server would most likely look like at the start. All cells would reside on one server. For example, we can begin deployment with a single

server. Now let's say our single server performance is becoming dismal, we can add a second server and offload several Cells. This offloading of Cells can continue until saturation, i.e., all Cells are running on their own server. This is Distributed Cell Architecture in a nutshell!