Disaster Recovery Server Design Specification

Revision 0.3.0 (DRAFT IN PROGRESS)

July 5, 2010

Prepared by: **GroundWork Open Source, Inc.**
139 Townsend Street, Suite 500
San Francisco, CA 94107

Contents

Tables

Figures

Document Restructuring

Tear this paper apart and reassemble it in combination with text now residing in other files, to create the following separate documents:

DR Sales Slides
These slides describe the driving rationale, concepts, very-high-level flow diagrams, features, and benefits — making the case for DR as a significant capability beyond the scope of a standard GroundWork Monitor installation.

DR Operations Training Slides
These slides describe what the GroundWork Operations and Support personnel need to know.

DR Concepts
This white paper is for general consumption, to be sent to customers who ask for more detail about our DR capability because they need more convincing than just the DR Sales Slides will provide, before making the purchase decision. This paper covers concepts, general architecture, narrative description, and system requirements. All of this is covered as a high-level overview, focusing on features, benefits, and major operational concerns.

DR Implementation Guide
This paper is for use by GroundWork Operations staff during installation, and for Customers who have already purchased the software, to assist their staff during their own internal training, policy and procedures development, and subsequent day-to-day operation. It must include exactly detailed installation and operating instructions, including specific sections that describe the Customer responsibilities during installation and operation phases.

DR Technology Description
This paper is a GroundWork-internal discussion of what we did and did not build for the initial DR prototype, and why, including the chosen architecture and implementation framework, file handling structures and protocols, database handling, test procedures, future product features, and an in-depth technical discussion of alternatives.

1 Overview

GroundWork has long provided a Standby Server capability, but it does not fully meet the needs of many customers in a true Disaster Recovery (DR) scenario. A principal difference is that the secondary server should not just be able to take over principal responsibility for monitoring, but also become the master source for subsequent configuration changes until the Primary Server is back in operation. And then whatever changes were made on the secondary server must be synchronized back onto the Primary Server so it can resume this duty.

This document is a place to record details of specific market and internal requirements to support such a configuration, and to propose a design to cover this space. Certain portions of the text in this document have been partly borrowed from the previous GroundWork *Standby Server Best Practices* white paper, with significant revisions to reflect changes to provide a more robust DR setup.

1.1 Customer Requirements

The following items have been agreed to for delivery of this solution to the first customer. The text here reflects that in the associated Change Order, and as such refers to a Standby server when in fact we would now classify this as a DR server. This terminology will be modified in later chapters.

1.1.1 Standby Monitoring Coverage Requirements

The Standby system must provide the functionality of the following subcomponents in the event of a Primary system failure.

- Perform active Nagios polling of configured devices and services.
- Perform active Cacti polling of configured devices and services.
- Provide the same SNMP Trap processing capabilities as the Primary.
- Provide the same SYSLOG message processing as the Primary.
- The Standby must reflect the runtime states of the Primary for scheduled downtime, comments, and external commands.

1.1.2 Standby Notification Requirements

In the event of primary server/system failure, the Standby server/system must provide the same notifications as would have been sent by the Primary system.

1.1.3 Standby Operator Interface Requirements

The Standby system must provide the same functionality as the Primary in the event of a Primary system failure. This functionality includes at a minimum the following interface elements:

- Dashboards
- My GroundWork (Dashboards)
- Event Console
- Status Viewer
- Reports
- Nagios
- NMS

1.1.4 Standby Administrator Interface Requirements

The Standby system must provide the following administrator user interface elements in the event of a Primary system failure:[1]

- My GroundWork (configuration)
- Auto Discovery
- Administration
- NMS

1.1.5 General Requirements

The proposed modifications are intended to operate on Primary and Standby systems in separate data centers over high latency links.

Securing communications between the Primary and the Standby server is possible via a number of commercially available methods and is beyond the scope of this document or the proposed modifications. Those protocols are as follows:

- Database replication via MySQL socket communication TCP/3306
- Nagios Service Check Acceptor (run-time updates for acknowledgments and downtime scheduling, etc.) protocol with DES encryption TCP/5677[2]
- MySQL snapshot replication via SSH TCP/22
- Flat file replication via SSH TCP/22
- Control signals via SSH TCP/22
- Heartbeat signals via unencrypted TCP socket communication TCP/5699

The Standby system will assume all described functions of the Primary system within 10 minutes of detecting a Primary system failure. A network split or any other communication problem between the Primary and Standby is defined as a Primary system failure in addition to other known failure modes.

1.2 Technical Requirements

Certain capabilities are needed to support the stated Customer Requirements.

1.2.1 Supported Configurations

The first customer's configuration involves only a Primary server, with no associated Child servers. The NMS add-on product will run on the Primary server.

1.2.2 General Requirements

Methods which synchronize data between Primary and Standby servers must take into account the fact that the Primary copy of individual pools of data may be subject to modification at any time. This could mean that the data is in a state of partial change at the time the synchronization method comes around to work its magic. It is therefore necessary that the synchronization mechanism either be able to detect incomplete sets of changes, so it can avoid propagating an inconsistent or broken configuration to the other Server, or allow for rollback if such a configuration gets installed.

1.3 Terminology

We use the term DR Server instead of Standby server to clearly distinguish the different classes of functionality, and the extended reliability support provided at the DR level. See the Glossary for a more extensive list of terminology used in this document.

1. Use of Standby Administrator user interface elements implies that the system has assumed the role of Primary and has become the authoritative source for configuration information. To prevent configuration change collisions, the system operator must take measures to ensure that Primary to Standby replication is disabled. Replication of modified configuration information on the Standby to the recovered Primary system is a manual process that needs to be performed by the system operator. GroundWork will provide instructions on how this is achieved.
2. Note that the original customer Change Order listed this port as 5668. The actual default port we use for the Bronx Command Acceptor port is 5677 instead, so that is what we are specifying here.

2 Background

Historically, GroundWork has supported a Standby server, with fairly limited capabilities. Here we wish to develop a deeper understanding of types of distributed installations and their associated reliability, with an eye toward defining a separate product function not previously available.

2.1 Categories of Distributed Operation

Different types of distributed operation reside in an overall Availability Space, which can be diagrammed as shown below. (An additional dimension of distributed operation would be "Scalability", but that generally addresses capacity more than availability, and so we will ignore it in the present discussion.) The best description of these two axes that I have seen is in [DR Slides] (see the References).

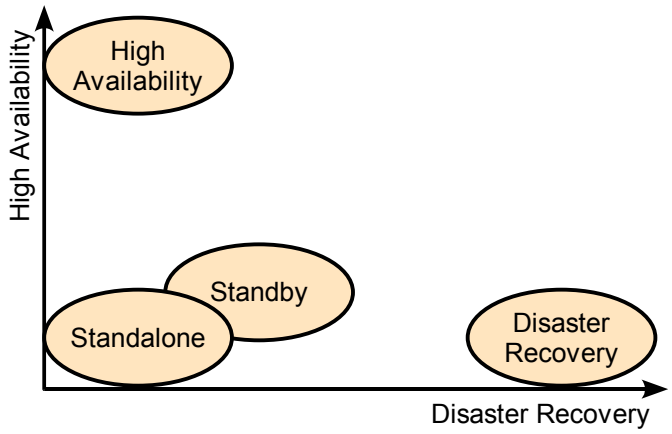

Figure 1: Availability Space

The axes in this graph represent different types of focus with regard to providing redundancy in a backup server.

Standalone Server

This type of availability just provides the original server, with no redundancy except that which might be provided on a single server (e.g., RAID storage).

Disaster Recovery

This type of availability provides the ability for a secondary server to assume full permanent responsibility for all aspects of monitoring. Key features distinguishing this configuration from a Standby setup include:

- Fully automated transfer of notification capabilities to the secondary server when the link to the Primary server is broken (*but wait, Standby Server seems to have that already*).
- Formal transfer of Master Configuration Authority to the secondary server during the Primary server outage. This capability involves a few simple manual steps to prevent problems when Primary/DR connectivity is intermittent, and to avoid administrator confusion as to where Master Configuration Authority resides at any given time.
- Dashboards, user-authentication and authorization data, and other site configuration are replicated on the DR server so they are available there should a failover event occur. This includes both syslog-ng logging and SNMP trap handling.
- Certain settings dynamically configured on the Primary server, such as downtime scheduling, operator comments, and external commands, are immediately forwarded to the DR server instead of waiting for a periodic synchronization point.
- Both failover and switchback scenarios are considered in the design and cleanly handled, so that (for instance) configuration changes made while the system is operating in DR mode get properly reflected back to the Primary server when it comes back up.

High Availability

This type of availability would provide completely automated and transparent transfer of all monitoring and administrative functions between the Primary and secondary servers, making all transitions seamless to both the end users and the administrators. Performance data such as RRD metrics might be time- and metric-synchronized between Primary and HA servers.

Due to the intense development requirements of such a solution and the limited number of customers actually needing it, GroundWork is not investing in this level of availability at the present time.

Standby Server

This type of availability provides the ability for a secondary server to assume temporary responsibility for basic monitoring. This provides coverage for cases such as regularly scheduled maintenance downtime on a Primary server, or for short-term unanticipated power outages on the Primary setup.

- Monitoring data collection and notification capabilities are assumed by the Standby server while it is in operation when the Primary server is unavailable.
- Assumption of Notification authority by the Standby server is automated when a Primary system failure is detected.
- Some capabilities, such as dashboards, Master Configuration Authority, configuration of syslog-ng, SNMP traps, etc. are not automatically available on the Standby server. Passive alerts such syslog messages and SNMP traps may be broken in Standby mode if source devices can only send to one destination and remain pointing to the Primary server. On the other hand, in some deployments where full business continuity is not the issue, and when the expected time to repair or replace the primary system is within reasonable bounds, such limitations may represent an acceptable level of service degradation during a short period when staff efforts are mostly concentrated on bringing the Primary system back into operation.

This is historically the level at which GroundWork has provided extended availability in enterprise deployments, following an 80/20 strategy. It has some degree of overlap with both DR and HA capabilities.

- It is like a DR system in that it provides an alternative system to run the monitoring if the Primary system fails. (However, unless the Standby server is located in a separate data center, that capability may be lost along with the Primary server. Also, replication of the Primary configuration is not nearly as automated as with a true DR solution, Master Configuration Authority is never transferred to the Standby system, and as such, switchback capabilities are not provided.)
- It is like an HA system in that a hot Standby should provide continuous uptime across a Primary system failure. (However, transition to the Standby is not seamless, and monitoring data on the Standby system will not be point-for-point identical with that of the Primary system.)

2.2 Monitoring Configurations

A full product solution must address a variety of possible base monitoring configurations, such as:

- GroundWork Monitor Primary server (only)
- GroundWork Monitor Primary server plus NMS
- GroundWork Monitor Primary server plus Child server(s)
- GroundWork Monitor Primary server plus Child server(s) plus NMS
- various distributed configurations (often, to support geographically distributed sites)

2.3 Excluded Capabilities

For clarity of thinking, we define here certain possible capabilities that we are explicitly not addressing.

n+1 Child Monitoring Servers

The present paper considers how to design a system with either non-redundant Child servers, or possibly with fully-redundant Child servers, but not with possible more-intricate setups involving partially-redundant Child-server setups.

3 Solution Overview

In this chapter, we discuss how the proposed solution operates.

3.1 Key Concepts

The formalization of certain critical ideas is key to distinguishing the DR Server capability from the older Standby Server model of providing redundancy.

3.1.1 Notification Authority

At any given time, either the Primary Server, or the DR Server, or both, may have Notification Authority assigned to it. This property gives it permission to send notifications from the ordinary GroundWork Monitor infrastructure sensing activities. Such notifications will include events such as alternate-server failure or link failure, issues which are directly relevant to operation of the DR setup, in addition to ordinary problems in the rest of the monitored infrastructure.

In order to assure that the system administrators are timely informed of all failures, Notification Authority is usually grabbed and released automatically by the DR server. This provides the necessary functional backup while avoiding unnecessary duplicate notifications in normal operation. However, Notification Authority is also subject to a manual override, which can be used to lock such authority in an enabled state. This override may be useful when the overall setup is operating in failure mode, with the DR server assuming continuous principal responsibility for monitoring even though the Primary server can sometimes be contacted and appear healthy. Notification Authority can also be locked in a disabled state, which may be appropriate (for instance) to block notifications while maintenance activities occur during scheduled downtime on the Primary system.

Forcibly disabling the acquisition of Notification Authority is potentially dangerous in the sense that one would never want both Primary and DR systems to be concurrently so disabled, perhaps by mistake. However, such hobbling can be necessary during installation and maintenance activities, for instance during software upgrades, and it is therefore allowed in these particular modes of operation.

[*Describe having Notification Authority vs. controlling Notification Authority. Describe the 3x3 matrix of local and remote control states (grabbed, dynamic, released).*]

Notification Authority is controlled by these commands:

Table 1: Notification Authority Control Commands

Command	Meaning
notify grab	arrogate Notification Authority, and lock this choice
notify dynamic	allow the choice to float, based on current conditions
notify release	relinquish Notification Authority, and lock this choice

We have these possible states of Notification Authority Control on the local and remote systems, where for purposes of the state descriptions we assume that Local is the Primary site and Remote is the Disaster Recovery site:

Table 2: Possible Combinations of Notification Authority Control

		Remote Notification Authority Control		
		Grabbed	**Dynamic**	**Released**
Local Notification Authority Control	**Grabbed**	Conflict, as both sides will generate notifications. Warn when this state is established, and during every heartbeat analysis using current remote state that finds this condition, about duplicate notifications.	OK (can be used as setup for normal-mode operation; allows both failover and failback of Notification Authority to occur).	OK and useful under some conditions, but failover can no longer occur; warn about that.
	Dynamic	OK (can be used as setup during long-term failure-mode operation; allows failback to occur when normal mode resumes, though should this setup then persist, duplicate notifications would be generated).	If the heartbeat analysis logic were too simplistic, this combination would potentially be dangerous, because the system might be left with neither system in charge (notifications could go missing or could be duplicated). To limit the possible damage, a normal-mode heartbeat analysis using current remote state will enable notifications only if the local system has Master Configuration Authority and the remote system does not, as a kind of polite deference to the Primary system. (If both systems have or neither system has Notification Authority, we do enable notifications on both sides.) With such a deference rule in place, this combination of control states becomes the standard mode of operation, reducing the need for the administrator to ever grab or release control. (If not for the deference rule, during normal-mode operation, we would need to generate a console warning if this condition were detected during a heartbeat analysis using current remote state, and always enable notifications on both sides, even in normal mode. That would mean duplicate notifications while this condition holds.)	Potentially dangerous, as the system may be left with neither system in charge (notifications may go missing) when failure mode is triggered. With this setup, a heartbeat analysis on the Local system (only) using current remote state will always enable notifications (though simply because the Local system has dynamic Notification Authority Control and Master Configuration Authority, not because of any logic involving the Remote system's Notification Authority Control setting).
	Released	OK and useful under some conditions, but failback can no longer occur; warn about that.	Potentially dangerous, because if the heartbeat analysis logic were too simplistic, the system might be left with neither system in charge (notifications could go missing). To prevent that from happening, a heartbeat analysis on the Remote system (only) using current remote state (i.e., current Local state) will always enable notifications in both normal and failure modes. This is not considered an ideal operating configuration (having the DR system handle all notifications even while operating in normal mode). But it could be used, for instance, if the equipment used for notifications from the Primary system were inoperative. More likely, you would want the DR system to have grabbed rather than dynamic control in such a situation, to make the intention clear.	Dangerous, as no notifications will ever be generated; warn during every heartbeat analysis using current remote state that finds this condition, about having no notifications sent from either side.

[*Note that we should also consider the possibility that the Primary and/or DR system may be operating in a NOC environment, wherein operators are stationed at the GroundWork Monitor system for at least certain work shifts and might not want notifications generated during those shifts, as they may depend instead on watching the Event Console and other screens to be informed of system failures. Thus a future product feature might involve automatically disabling notifications at certain sites during certain configured time periods. We would not only want Notification Authority Control to be automatically*

released during such periods, we would also want the usual heartbeat-analysis warnings about such a configuration to be suppressed so the Event Console is not inundated with them. The same may be true during a scheduled maintenance period, when Notification Authority Control is intentionally released because the system is considered to be in an unstable or incomplete-setup state while maintenance operations are proceeding (though Monitoring and DR might both be operating during part of this period).]

3.1.2 Master Configuration Authority

Master Configuration Authority is permission to make changes to the setup of the system. In contrast to Notification Authority, Master Configuration Authority is never assumed automatically; it must be manually assigned. This is done by design, both to ensure that the administrators know explicitly on which system changes must be entered, and to ensure that such authority is never unceremoniously ripped away by the machines while the administrators are in the process of entering changes.

In normal-mode operation, Master Configuration Authority resides in the Primary Server. In failure mode, when the DR Server has assumed Notification Authority, Master Configuration Authority may remain assigned to the Primary Server for some time, possibly throughout an entire outage episode, if the customer believes the disruption will be of limited or controlled duration. This can happen, for instance, during a regularly scheduled biweekly downtime for the Primary Server for ordinary maintenance (e.g., application of OS-level patches, replacement of hardware components, network rewiring, and so forth).

Because of the manual assignment process, at any given moment the Primary system, the DR system, both, or neither may have Master Configuration Authority. This is not generally disruptive to operating the systems unless both systems currently have this authority assigned. In failure mode, for instance, if the Primary system is not functioning to the point where said authority can be formally disabled there, it may be enabled on the DR system so the administrators can continue to perform their duties. In this situation, both Primary and DR systems will have Master Configuration Authority, at least temporarily. If the systems can contact each other and discover this fact, notifications will be sent from both sides so the situation is brought to the site's attention and be corrected. In the meantime, replication between the two servers will be disabled, as there will be no clear principal source of changes.

A system with Master Configuration Authority becomes the source of change information when the setup is replicated between servers. A system without it becomes a sink of change information. Care must be taken during transitions of Master Configuration Authority to ensure that both ends of the connection are properly synchronized before the direction of change data is reversed, lest recent changes on the new receiving end be overlooked and lost during subsequent synchronization.

3.1.3 Staged Mirroring

[*The text in this section is just a quick note to get the concept registered. It will be revised.*]

[*insert figure here*]

A critical part of keeping a Disaster Recovery system available is to keep its own configuration consistent and useable. And that means we cannot allow replicated files to exist in a partially-complete state, lest they be called into duty at an inopportune time.

The solution is to used staged mirroring. File /path/to/file on the Master system will not be replicated directly to /path/to/file on the Slave system. Rather, mirroring will occur to a pending-configuration tree. And only when the configuration is believed to be complete and ready to use will it be switched into use on the Slave system.

In the DR prototype, this belief will be established at the end of each heartbeat cycle. That's not actually a perfect solution, because it has to do with time and completeness of available changes rather than a confirmation from each individual application that its altered-so-far configuration is actually valid. A future product version of DR might interact with applications to perform such validation.

In addition, because of the possibility of failure when installing altered configurations, especially without full prior validation by all the affected applications, the existing configuration will not simply be overwritten with new files. Rather, when a DR heartbeat operation needs to copy in altered files, it will first freeze the existing setup in a timestamped backup, so it will be possible to easily roll back to an earlier configuration should the need arise. The local DR control UI will allow the administrator to list the available backups and roll back to a selected setup with just a few commands.

[*Insert figure here showing the structure of the staged mirroring file trees: pending (mirrored-to from the opposing server), ready (complete and available for rolling into production), working copies, shadow (in the process of being rolled out of the working copy), and timestamped backups. Show the dataflows between these copies.*]

3.1.4 Database Replication

There are two important repositories of configuration data that the DR system must protect — data stored in databases, and data stored elsewhere, typically in flat files. The latter will be discussed below. Here we describe how data in databases is replicated.

To ensure the most up-to-date capture of configuration changes, the DR model in this design will use automatic, continuous database replication between master and slave databases whenever possible. A system with Master Configuration Authority will serve as the source for changes to critical databases and individual tables, while a system without said authority will instead continually process new entries in the transaction log from the master database and reflect certain of those changes in its own databases and tables.

This model of operation will keep both Primary and DR systems synchronized continuously while both systems are up and running, with a slight lag time for changes to propagate. Should the link between these systems be temporarily broken, or should the DR system be taken down for its own scheduled maintenance or because of an unexpected outage at the DR site, changes in the Primary system database will accumulate and be loaded into the DR database once it returns to service.

In the case of extreme desynchronization, such as replacement of a failed disk drive on the DR Server, playback of the transaction log may be infeasible, and a dump/restore mechanism may be substituted to initially bring the systems back into orderly concurrence. For that reason, even for databases where continuous replication is used, periodic snapshots will also be taken to provide a fallback safety net.

For the slave system to continually and incrementally read the master database transaction logs, they must be visible in the filesystem of the slave. For this reason, NFS must be used to export the logs so they can be remotely read. In these days of heightened security awareness, especially given that a DR system will probably be sited at a remote location and connected via long wires, Secure NFS will typically be used to protect access to the data. Among other things, this requires that the servers at both ends be time-synchronized. Details of Secure NFS installation and maintenance are outside the scope of this document, but form an important part of the initial DR setup process.[3]

Certain technical assumptions (e.g., adequate disk and network performance) underly this design choice. These issues are discussed further in later chapters. Also, correct handling of the database transaction logs becomes a key consideration when Master Configuration Authority is transferred between systems.

Some situations render the use of database-level continuous replication impossible. Most importantly, databases which are managed by a long-running cache, such as the Hibernate persistence framework, cannot be so replicated. The cache on a receiving system would not be aware of changes made directly to the underlying database, which would cause considerable confusion and probably collisions as two different sources (the replication process and the cache) vie for control of the stored data. Therefore, for such databases, we must take an alternate approach:

- freeze the source database at a transaction boundary
- make an external copy of the database
- unfreeze the source database
- transfer the copy to the sink server
- pause the sink cache (which probably means stopping it completely) and probably applications that depend on it
- installing the copied database
- restart the sink cache in such a way that it repopulates itself from the modified content

3. The use of other shared Linux filesystems is also possible, but has not been evaluated to date. See, for example:
   - DRBD (http://en.wikipedia.org/wiki/DRBD)
   - GFS2 (http://en.wikipedia.org/wiki/Global_File_System)
   - NBD (http://en.wikipedia.org/wiki/Network_block_device)
   - OCFS2 (http://en.wikipedia.org/wiki/OCFS2)

   Some of these choices may be more appropriate for a local cluster than for widely separated servers. Note especially that DR may be implemented via a low-available-bandwidth connection between widely separated servers (e.g., located on East and West coasts).

This process means that synchronization will occur at periodic points rather than continuously, for the affected databases. The meaning of Master Configuration Authority will still apply in this context, identifying which system is serving as source and which as sink for replicated data.

3.1.5 Static Configuration Replication

[*This section partially overlaps what is in the previous section. Revision will occur.*]

[*Talk about scheduling the replication, as well.*]

Data stored in databases is often changed in atomic transactions, which either commit an entire set of related changes or roll back whatever partial changes may have been made from the set.[4] In contrast, data stored in files typically has no formal mechanism for recognizing transaction boundaries. Faced with this situation and the concomitant need to preserve a working configuration on the DR system because Primary Server failure could occur at any time, the DR design must provide for datafile replication to occur in a carefully controlled dance, so the DR system can quickly take over principal responsibilities even if it was in the middle of accepting updates when a failure occurred.

To provide that capability, replication of datafiles must be done as a series of stages.

1. The source applications must be quiesced so the source files are known to be in a consistent state.
2. A collection of related files is copied to a local repository, so the set is known to be complete before any transfers are made to the slave system.
3. The set of files is transferred from master to slave system, and tagged as a related set.
4. When all source files related to an application have been transferred, or when such transfer is seen to fail, the application will be activated so it can resume normal operation if it was previously running.
5. The receiving end is instructed to install the new files. This replacement action must occur entirely under control of the receiving end, without further interaction from the source end, because link failure might occur at any time during the installation process.
6. When the receiving end has completed its installation of a set of files, it returns status to the source end so further replication can continue.

The kinds of actions needed to quiesce and resume active operation, the identification of sets of related datafiles, and any required installation operations all require a potentially complex configuration setup of the replication itself. The details of this configuration will vary from application to application whose files are being replicated. The replication configuration itself becomes part of what must be replicated between the Primary and DR servers.

3.1.6 Replicated-Data Transformations

In some cases, configuration data cannot be used as-is when copied directly from master to slave system. For instance, different network paths will be used for monitoring from the perspective of the Primary and DR systems, and corresponding monitoring dependencies must be adjusted to match whenever a new configuration is installed. Also, notification channels used by the Primary and DR systems should be different to avoid common-mode failures causing the suppression of all notifications. Some differences like these must be explicitly, manually configured, while others should or must be modified on-the-fly when the configuration is replicated and installed for use on a slave server.

Automated changes can be a tricky process to design, because the nature and extent of such transformations may not be immediately apparent. In addition, while such transformations may be relatively simple to carry out on collections of static files, automated database replication may or may not provide the necessary hooks for transforming portions of the master configuration data as it is being reflected into the slave database. The situation can be even more complex if database changes are made on the master without the modifying application necessarily grouping them as transactions to preserve required relationships throughout every update.

Specific situations we expect to encounter must be dealt with individually, and will be detailed in the installation phase of the DR setup.

4. Transactions are useful and common but are not enforced by the database, which itself generally has no idea which sets of changes are conceptually related at the application level. Some applications may save configuration changes incrementally in the database without treating them as formal transactions. Master and slave systems which use such data must be aware that at any given time, the configuration data could be in an inconsistent state requiring some level of validation and repair before use.

3.1.7 System DR States

Given the ideas above, it becomes important during operation to understand the state of the systems, so any human interaction or control of the systems is done with knowledge of the current context. To that end, the following substates are defined and must be made visible to the administrator:

has Notification Authority
The system (Primary or DR) currently has notifications enabled.

has Master Configuration Authority
The system (Primary or DR) currently is allowed to accept configuration changes from administrators.

replication is enabled
Replication between Primary and DR systems is currently allowed. It must be allowed on both ends for replication to actually proceed. It will generally be disabled during downtime for system maintenance activities, whether regularly scheduled or not.

replication operations are currently active
Replication actions are currently occurring. This can refer either to replication of static configuration data in ordinary files, or to wholesale database updates. It will not refer to database replication that operates on a continuous rather than macroscopically periodic basis.

is quiesced for source replication
A certain component is being held in a safe state, possibly temporarily inoperative, while source data is being captured for replication to the sink.

is quiesced for sink replication
A certain component is being held in a safe state, possibly temporarily inoperative, while already-transferred data from the source is being installed on the sink.

is out of sync
Comparable components on the master and slave systems are known to not be synchronized, past the brief ordinary time lags involved as continuous or periodic sync operations occur. Recovery of a fully synchronized state may involve wholesale rather than incremental transfer of configuration data for the affected components.

3.2 High-Level Architectural Model

The diagram below shows the data and control paths between the major system components. A much more detailed illustration appears in an appendix, but this initial level of understanding should be adequate for the following introductory discussion.

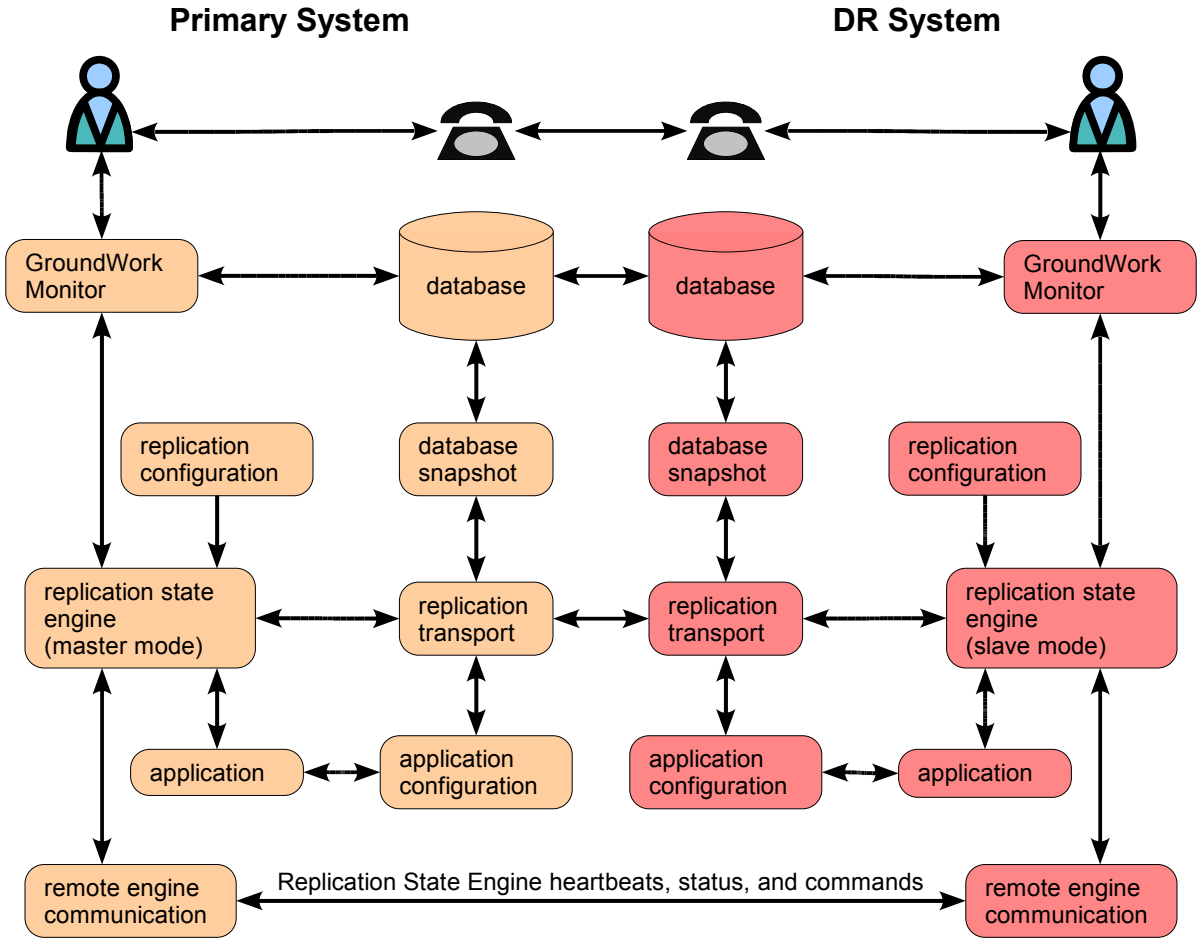

Figure 2: High-Level Replication Architecture

3.3 Major Components

application

xxx

application configuration

xxx

databases

[*explain differences in tenor and handling between monarch, jbossportal, GWCollageDB, cacti, and nedi*]

database snapshot

xxx

DR heartbeats

Regular communication between Primary and DR systems is essential for triggering automatic assumption of Notification Authority by the DR system when the Primary system is down or inaccessible. The heartbeat messages contain more than just system health information, though. They also carry information about the system state at each end, so the replication state engine is

always aware of whether the remote system has replication enabled and has Master Configuration Authority. This state information is used, for instance, to trigger notifications at each end if both ends are discovered to have Master Configuration Authority.

GroundWork Monitor
The standard GroundWork Monitor Enterprise Edition product is installed on each system, and configured to support the DR setup (cross-monitoring, notification targets and paths at each end, etc.). Also included in this general component is the NMS add-on product.

people and phones
Not everything can or should be automated. Since the network link between Primary and DR systems may fail, coordination via other channels will be necessary during an extended DR event. This also implies that some level of formal procedure should be established at the customer sites prior to actual problems, to be followed during upsets.

remote heartbeat sensor
xxx

replication configuration
xxx

replication state engine
The central point of control for all configuration data copying and installation resides in the Replication State Engine, an instance of which resides on each system. This engine interprets the DR heartbeats, manages the System DR States noted above, and instigates replication operations as needed. It is itself controlled both by static replication configuration data (which itself becomes part of the replicated data), and by human-initiated commands to change the system state. In turn, it is responsible for dynamically arrogating and relinquishing Notification Authority by enabling and disabling notifications in its associated local copy of GroundWork Monitor.

replication transport
Changes to static configuration data, and sometimes whole-database snapshots as well, must be copied between systems via some mechanism. This facility is abstracted to allow the replication state engine to deal with data copying at a higher level, and to allow the transport implementation to change between sites and over time. A typical implementation might utilize `rsync` (http://samba.org/rsync/) through an `ssh` tunnel.

3.4 Narrative Description of Operation

At any given time, the overall system runs in one of four operating modes. These modes cover the life cycle of the system:

- Installation Mode describes how the system is normally installed and configured.
- Normal Mode describes how the system works in day-to-day operation.
- Maintenance Mode describes how the system configuration is updated with additions, deletions, and coverage changes during normal operation.
- Failure Mode describes how the system responds to various failures.

The following subsections cover all four modes for the DR system.

3.4.1 Installation Mode

The Primary and DR Servers are installed with identical standard GroundWork Monitor Enterprise Edition product software. The Monarch subsystem on the Primary Server is used to configure both the Primary and DR Servers. The Monarch subsystem on the DR Server is not normally used, as its complete configuration setup will be managed from the Primary Server. (Only when the roles of Primary and DR servers are switched will the DR system become the master for configuration changes.)

The DR software package is installed on top of GW Monitor EE to augment the base-product capabilities and provide the necessary heartbeat, replication, notification control, and failover/switchback mechanisms.

Each system that is sending SNMP traps or syslog messages to the initial Primary Monitoring system must be reconfigured to send the trap and syslog messages to both Primary and DR systems. If any

systems cannot be configured to send to two destinations, it is recommended that a virtual IP addressing scheme be used to deliver the events to both systems.[5]

[*The complete installation procedure is TBD, but will be based on an RPM plus some local configuration changes.*]

GroundWork recommends that the DR Server be installed after the Primary Server has been installed and calibrated in all respects. This provides a known-stable setup before the additional complexities of DR support are considered and implemented. The required DR software will be available through GroundWork Support.[6]

3.4.2 Normal Mode

During normal operation, the Primary Server processes any alarms it detects and generates the notifications that have been configured. The Primary Server also monitors the DR Server, including the link to the DR system, the DR server machine itself, and the health of the GroundWork Monitor product. Alarms and notifications are provided if any of these aspects of the DR Server should fail.

THe DR Server is fully operational during normal mode. It performs all the same service checks performed by the Primary Server, although from a different network perspective. That means that some reporting dependencies are likely to be different between the Primary Server and the DR Server setups. The DR Server's monitoring actions include active service checks, passive service checks, trap processing, and logfile processing. The DR Server participates in DR heartbeat exchanges with the Primary Server, usually run once each minute. These heartbeats tell each side not only whether the other side is running, but also the state of the remote system (e.g., whether it currently has Notification Authority and Master Configuration Authority) and whether configuration changes are available. In normal mode, the DR Server runs without Notification Authority and thus has its notifications configured off.

3.4.3 Maintenance Mode

Additions, deletions, and changes to the configuration of the Primary and DR Servers are made as follows.

Usually, the Primary Server has Master Configuration Authority, and the DR Server operates without it. In this case, changes to the GroundWork Nagios configurations of both systems are made using the Monarch instance residing on the Primary Server. Failover and switchback scripts in the DR package are installed on both the Primary and DR Servers so that control can be transferred in either direction. These scripts are used with event handlers to convert the DR Server to assume all Primary Server functions, or to have the Primary Server resume its usual predominance.

Changes to configurations for the SNMP trap and syslog handlers are made on the Primary Server and then are automatically copied to the DR Server. Database and logfile maintenance, such as backups and logfile rotation, must be made separately on each server. Some of these operations might be only manually initiated, such as database backups, while other operations such as message-table pruning, database index optimization, and logfile rotation may run as regularly scheduled automated scripts. Changes to the configuration of other tools, such as the NMS add-on package, are also made on the Primary Server and allowed to percolate through the automated replication processes to the DR Server.

Most transitions between Normal Mode and Maintenance Mode are automated, with the replication state engine taking care of quiescing and restarting components on the Primary and DR systems as needed to copy configuration changes between servers.

3.4.4 Failure Mode

Apparent failure of the DR Server (the link to it, or the server machine, or the GroundWork Monitor product) results in alarm generation and notifications by the Primary Server. In this sense, the DR Server is like any other monitored host in the infrastructure.

5. Failing that, the residual asymmetry of the monitoring configuration on the Primary and DR systems might provoke a need for data transformations when the configuration is replicated. This possibility remains to be explored.

6. I put that statement in as a placeholder. Someday when this package gets formalized into a product, formal licensing and access through the GroundWork Connect Support Portal should perhaps be allowed, based on the customer's account having licensed this add-on product.

Each side of the server pair uses monitoring and the DR heartbeats to assess the state of the other server. A DR heartbeat will typically be run every minute. As long as the DR Server can verify that the Primary Server is running normally, it will remain quiesced with respect to generating notifications. When the state of the Primary Server cannot be so verified, the DR Server waits for several heartbeat cycles before changing its own behavior, to ensure that the failure was not just a transient phenomenon. If the failure persists or if it comes and goes repeatedly, the DR Server assumes that the Primary Server may be down, and this circumstance causes the DR Server to arrogate authority to send notifications. One of the first notifications sent from the DR Server will typically be about the failure of the Primary Server or the link to it. Notification Authority will be automatically relinquished after a stabilization period if the Primary Server is once again seen to be up and running without further failure for a preconfigured time interval.

Immediately after a failover from the Primary Server to the DR Server, the DR system will be displaying accurate information from its perspective on state, events, and performance graphs, as it will have been monitoring all along and observing essentially the same state as the Primary Server. If the link is down and that link is used as part of monitoring, then of course by the time that complete failure is recognized and failover formally happens, a few heartbeats after the initial failure, some of the infrastructure is likely to be seen as inaccessible by the DR Server even while the Primary Server may still be able to see that same infrastructure.

Once the Primary Server returns to fully available service, as seen by the DR Server, a time lag will be imposed before the DR server fully believes the Primary Server will remain up and the DR Server should stop sending notifications. After relinquishing Notification Authority, in all other respects, the DR Server will continue to mirror the monitoring functions of the Primary Server.

After a Primary Server outage, the RRD and Foundation databases on the Primary can be expected to be missing data that would have been collected during the time between failure and restoration of the Primary. Inasmuch as the monitoring itself continues on the DR Server during this period, this gap is deemed to be an acceptable consequence of the failure of the Primary Server. The *Fault Tolerant Monitoring* white paper provides additional information about this design choice.

During a failover event, Master Configuration Authority is not conferred on the DR Server by default. It must be manually set by the system administrators so this change of state can be coordinated with the entry of configuration changes on the DR Server. If only the link is down, the DR Server might never be blessed with Master Configuration Authority during the entire episode. If the Primary Server is simply in a period of regularly scheduled maintenance, likewise the DR Server may remain untouched, with all configuration changes simply delayed until the Primary Server is back in operation. If the site does give Master Configuration Authority to the DR Server, this should be done with a clear understanding of any very recent configuration changes made on the Primary Server just before failover, so they can be verified or re-entered on the DR Server before making further changes there.

In effect, with different regimes normally in play for assumption of Notification Authority and Master Configuration Authority, there are four types of monitoring-control transition in play.

failover
: An automated cutover from the Primary Server to the DR Server, when a Primary Server failure is detected by the DR Server. This happens with Notification Authority.

failback
: An automated return of principal service to the Primary Server from the DR Server, when the DR Server is finally convinced that the Primary Server is once again up and stable. Again, this happens with Notification Authority, provided that the system administrators have not locked the DR Server into play as the principal source of notifications.

switchover
: A manual transition of power from the Primary Server to the DR Server. This happens with Master Configuration Authority, when the system administrators deem it necessary (i.e., when the Primary Server is expected to be out of play for an extended period during which configuration changes will be required).

switchback
: A manual return of power to the Primary Server from the DR Server. This happens with Master Configuration Authority sometime after a switchover, once the Primary Server is synchronized with changes made on the DR Server during its reign and ready to resume its original controlling duties.

3.5 Internal Design Structure

The preceding sections have provided an overview of the capabilities of the DR setup. In contrast, here we describe certain internal parts and behaviors chosen to implement those capabilities.

3.5.1 Database and Table Replication

The following databases and tables require replication to keep configuration data synchronized:

- `cacti`
  - Replicate this database only if NMS is installed.
  - (what is contained in this database?)
  - (what caching does Cacti do, if any, on the content in this database?) Answer: none, as the cacti process for polling is run periodically as a cron job. On the other hand, running it from the cron job on the slave server should be held off while the slave database is being updated. And as you would expect, the master database will need to be briefly locked while the source copy of the database is being frozen.
  - (think about that again. we want the configuration data in the cacti database to be replicated from the master machine. but intermixed with that data is the results of local cacti polling. so it is unlikely that we will be able to use continuous database replication at least unless we can separate these data types and only replicate tables containing only configuration data. and even if we cannot use continuous database replication, and need to do wholesale database replacement instead, we will need to be careful about what tables we replace, and to ensure that cacti is not operating when we do so.)
  - (which tables in this database should be replicated?)
  - Location and access credentials for this database can be found here:

    ```
    /usr/local/groundwork/enterprise/config/enterprise.properties
    ```
  - (what processes need to be bounced to pick up changes?)
- `monarch`
  - This database is the principal source for Nagios configuration data.
  - There should be no long-term application caching of the data in this database, other than perhaps in the `process_service_perfdata_file` script (the `datatype` and `host_service` tables). Hence there should be little difficulty in using direct database-level replication for this database, if we exclude those tables.
  - Replicate all tables in this database [except perhaps `datatype`, `host_service`, and `sessions`?].
  - [there may be some issues with transforming data in the `host_parent`, `host_dependency`, and `service_dependency` tables, to reflect the different network perspectives of the Primary and DR systems]
  - [there may be some issues with transforming Monarch Group definitions with respect to how the Primary and DR Servers are treated in such Groups]
  - Location and access credentials for this database can be found here:

    ```
    /usr/local/groundwork/config/db.properties
    ```
- `GWCollageDB`
  - This database stores a lot of dynamic monitoring state along with a small amount of locally-specified configuration data. There is also some derived configuration data stored here, driven by synchronization steps during a Monarch Commit operation. Only the locally-specified data, that which configures the Event Console, need be replicated to the slave server. The rest can either be derived from the slave server Monarch or (as with the `LogMessage` and `LogMessageProperty` tables) represents the dynamic state seen by the slave server itself.
  - This database is managed by Hibernate, and thus is not a candidate for continuous replication.
  - Replicate only a very few tables, primarily to support configuration of the Event Console.

        - GWCollageDB.Action
        - GWCollageDB.ActionParameter
        - GWCollageDB.ActionProperty
        - GWCollageDB.ActionType
        - GWCollageDB.ApplicationAction
        - GWCollageDB.ApplicationType
        - GWCollageDB.ConsolidationCriteria
        - GWCollageDB.OperationStatus
    - Location and access credentials for this database can be found here:

        ```
        /usr/local/groundwork/config/db.properties
        ```

- jbossportal
    - This database stores users, roles, portlet access permissions, and dashboard definitions and authorizations.
    - Replicate all tables in this database.
    - This database is managed by Hibernate, and thus is not a candidate for continuous replication.
    - (See the DR Operator Interface section in the appendices for further details.)
    - Location and access credentials for this database can be found here, though in a funky format that will need to be torn apart for use by the replicator:

        ```
        /usr/local/groundwork/foundation/container/webapps/jboss/portal-ds.xml
        ```

- nedi
    - Replicate this database only if NMS is installed.
    - (what is contained in this database?)
    - (what caching does NeDi do, if any, on the content in this database?)
    - (which tables in this database should be replicated?)
    - Location and access credentials for this database can be found here:

        ```
        /usr/local/groundwork/enterprise/config/enterprise.properties
        ```

    - (what processes need to be bounced to pick up changes?)

xxx

3.5.2 Replication Configuration

xxx

3.5.3 Atomic Operations

xxx

3.6 Automatic Actions

replication

failure sensing

failure notification

Notification Authority arrogation and relinquishing

3.7 Human Interactions

installation

configuration

maintenance

state changes

4 Solution Architecture

[This chapter is a placeholder for a more detailed description of the individual components and their behaviors and interactions. Some of the material which currently resides in appendices will migrate back to here. Probably the chapter name will change to be more descriptive of the eventual content.]

4.1 Dataflows

5 Solution Application

5.1 Local (Site) Design

supported configurations

5.2 Installation

OS-level installation: set up Secure NFS and ssh to share the database transaction logs between Primary and DR servers

5.3 Configuration

5.4 Operation

5.5 Events and Event Handling

5.6 Testing

(talk to Richard about what should go here)

5.7 Troubleshooting

TBD

Appendix A: Glossary

The following terms are used in this document.

backup mode
The condition of a server when it is running and not acting as the primary information processor, but instead is ready to take over primary duties should another server fail.

Child Server
A monitoring server which feeds data to a Parent Server and is not itself where configuration changes are managed.

Disaster Recovery (DR) Server
A server which is configured to take over all responsibilities of a Primary Server should that server either fail or become inaccessible.

failback
Automated transfer of control from a secondary server to the Primary server. While some definitions of DR assume this will be the usual method of placing the Primary server back in control, in our implementation we assume a switchback will be used instead, as this step requires coordination with the administrators who might still be using MCA on the DR server. See *failover*, *switchback*, and *switchover*.

failover
Automated transfer of control from the Primary server to a secondary server (be it Standby, DR, or HA). For instance, assumption of Notification Authority will be automatically triggered on the DR server by loss of the Primary server or the link to it. See *failback*, *switchback*, and *switchover*.

Failure Mode
A period when the system is responding to various failures.

High Availability (HA) Server
A monitoring server which provides essentially transparent failover/switchback operations in the presence of failures.

hot spare
A server running in backup mode that is continually processing information as a server in primary mode would do, but without the final authority to affect the outside world. It may be brought into play to replace a server previously operating in primary mode which has failed.

Installation Mode
The period when all parts of the system (both Primary and Backup servers) are normally installed and configured.

Maintenance Mode
A period when the system configuration is updated with additions, deletions, and coverage changes during normal operation.

Master Configuration Authority (MCA)
The designation of a server as being the single authoritative source for configuration changes. Updates to the configuration will be made on the server(s) so designated, and replicated to other servers on some periodic or dynamic basis.

Master Server
A server which is currently operating in primary mode with Master Configuration Authority. In normal operation, this will be the Primary Server, while during a DR situation, until authority is formally returned to the Primary Server, this will be the DR server. Configuration updates made on the Master Server will be reflected to the Slave Server to keep it up-to-date in preparation for a future transfer of Master Configuration Authority.

NMS Server
A server running GroundWork's Network Management Suite.

Normal Mode
A period when the system works in day-to-day operation, with the Primary server in charge of the monitoring.

Notification Authority (NA)
The designation of a server as being an authoritative source for generating notifications. This authority may sometimes be applied to both Primary and DR servers (say, when the link between them is down, and neither knows whether the other is able to send notifications).

Parent Server
A GroundWork Monitor server which accepts monitoring-data feeds from one or more Child Servers and manages their configurations as well as its own.

primary mode
The condition of a server when it is the principal actor, taking final responsibility for information processing.

Primary Server
The principal top-level server running GroundWork Monitor during normal, non-failover circumstances.

Replication Master
The server which is acting as a source of configuration data and controlling the replication operations.

Replication Slave
A server which is acting as a sink of configuration data and responding to commands from the Replication Master.

Slave Server
A server which is currently operating in backup mode, accepting configuration changes from other servers to prepare it for possible duty in the future as a Master Server.

Staging Server
A server on which software and configuration changes can be made and tested before rolling them out into production.

Standby Server
A secondary top-level server running GroundWork Monitor for use during Primary Server outages. This term refers to a backup server with certain functional limitations as described in the main text.

switchback
Manual transfer of control from a secondary server back to the Primary server. For instance, Master Configuration Authority will be resumed on the Primary server via a switchback after operating in Failure Mode. See *failback*, *failover*, and *switchover*.

switchover
Manual transfer of control from the Primary server to a secondary server. For instance, Master Configuration Authority will be transferred to the DR server via a switchover. See *failback*, *failover*, and *switchback*.

warm spare
A server running in backup mode that accepts configuration changes to keep it synchronized with another server for which it may take over monitoring duties, but which does not actively monitor until it assumes those duties.

Windows Child Server
A specialized Child Server which is configured to run batches of checks to monitor Windows machines, using Windows-specific APIs to probe those machines.

Windows Proxy Server
A primitive type of monitoring proxy that actively runs specialized checks remotely in a Windows environment and reports the results to a Parent Server. Its functionality is generally better managed by using a Windows Child Server instead.

Appendix B: Internal Design Details

The following figure shows the principal components, to be described individually below.

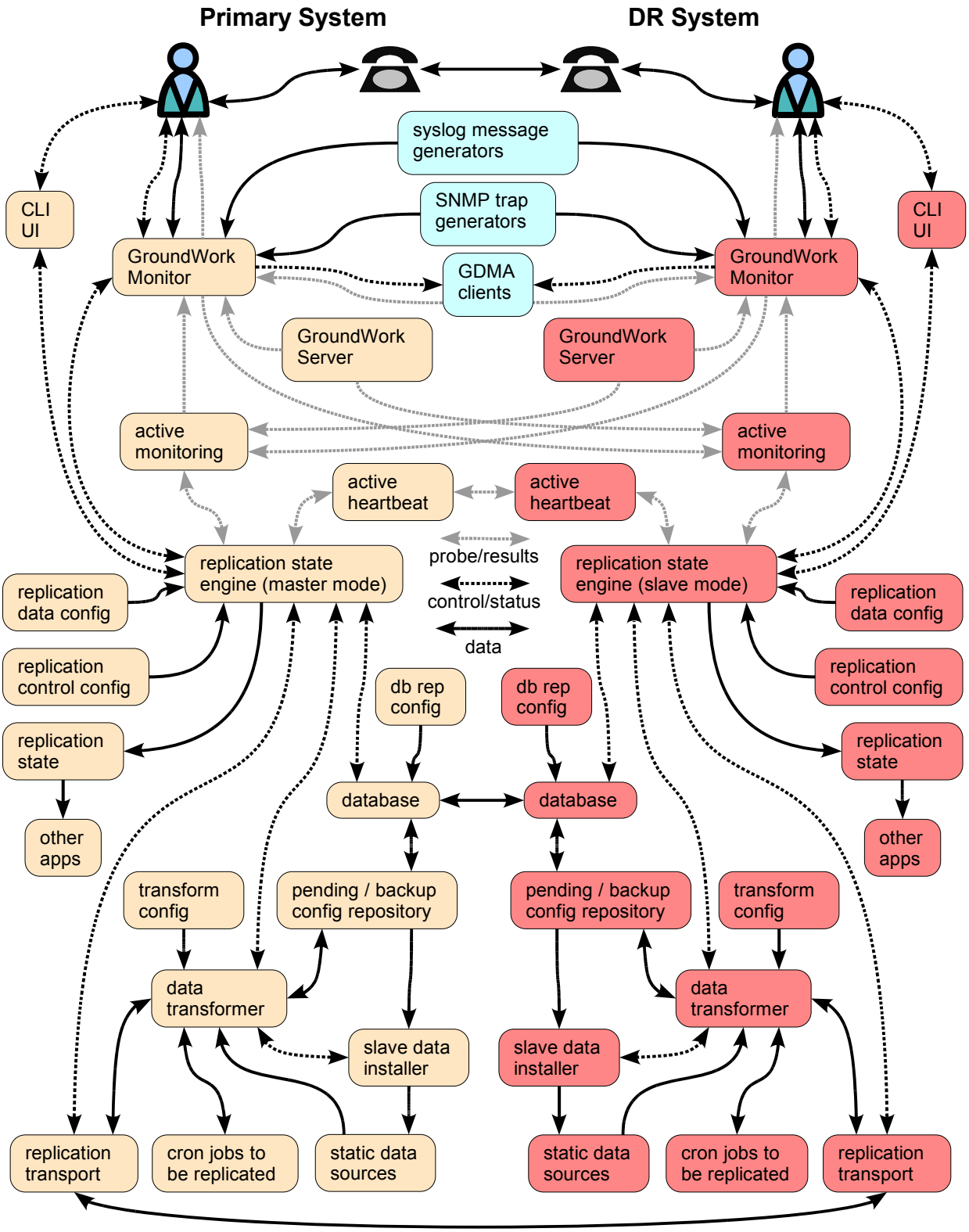

Figure 3: System Architecture: Components and Dataflows

B.1 Architecture

xxx

B.2 Components

Individual components shown in the System Architecture diagram above are listed here in alphabetic order to make them easier to find, though this does not correspond to their rough ordering within the figure.

[describe behaviors here, not just general functionality]

active heartbeat

active monitoring

CLI UI

cron jobs to be replicated

data transformer

database

db rep config

GDMA clients

GroundWork Monitor

GroundWork Server

humans
: The system administrators who manage and operate the monitoring systems at the Primary and DR sites.

other apps

pending / backup config repository

phones
: Used by people to communicate between Primary and DR sites, since the network link (and channels such as email) may be down. Used in particular when Master Configuration Authority is transferred between sites, so the critical administrators are informed as to who is allowed to make changes, and where.

replication control config

replicate data config

replication state

replication state engine

replication transport

slave data installer

static data sources

transform config

B.3 Dataflows

xxx

Appendix C: Distributed Replication Control Engine

To satisfy the needs of the DR configuration, we can either build a bunch of ad-hoc scripting for the various components involved, or formalize this functionality and build a centralized, supportable, extensible mechanism for dealing with this and related issues. Here we collect ideas for what a formalized capability would look like.

C.1 Features

- One system has Master Configuration Authority, and is therefore the source for all changes.
- Want a UI to control it:
  - display the current Master
  - relinquish, arrogate, or transfer Mastership
  - start/stop periodic replication
  - specify the replication source (from the Slave) and/or target (from the Master)
  - allow manual, selective triggering of particular synchronization operations
  - coordinate replications initiated from either Master or Slave, so as not to trample each other
  - schedule synchronization operations
  - possibly, schedule regular maintenance periods
  - trigger heartbeat probes and display their results
  - compare configurations between sites; detect and display differences
- negotiate Mastership:
  - transfer Mastership under manual control, using a full two-phase-commit handshake to do so reliably in the possible presence of link failures
  - detect violations of single Mastership, and generate notifications if such is found
- future extensions:
  - manage all configuration and associated-software backups, including tracking and restoration of such data (and locally-installed or modified scripts)
  - manage version control of configurations and local extensions or modifications
  - general: support a formal change-control / change-management process in a very convenient manner, perhaps by allowing a new configuration to be tested on a Staging Server before being put into production, then copied into production with certain limited and well-understood transformations (e.g., substitution of the monitoring-server hostname and possibly names of different child servers; substituting different data-collection dependency chains; substituting different notification targets and notification mechanisms to avoid common-mode failures) automatically made at that time throughout the installed configuration setup
- xxx

C.2 Components

This is the core of a DR product. It involves looking at the underlying ideas and getting them conceptually organized, before trying to create any kind of specific architecture to support a particular implementation of those ideas.

Key abstractions for DR functionality include:

- what types of data components need to be supported
  - cron jobs
  - static flat files
  - partial or entire file trees
  - entire databases
  - specific database tables
- what data components need to be transferred

- what transforms should be run on the data as it is transferred
- from where to where the data should be transferred
- on what schedule the data should be transferred
- what dependencies exist between transferred data components (ordering, coherent sets, ...)
- with what dependencies on running processes (that might need to be up or down for reliable transfer [data capture and replace] to occur)
- what procedures should be called to modify the states of processes
- under whose control the replication should proceed
- under what macroscopic conditions data should be transferred
- how those macroscopic conditions are to be sensed
- how to detect if data transfer fails
- what should be done if data transfer fails
- how heartbeats should work, and their configurability
- what (such as error conditions) should trigger actions, other than heartbeat failure (one particular fundamental error condition); a state-transition diagram of triggers and consequences (states, transitions, conditions, events, triggers, actions, alarms)
- how the package as a whole should be enabled and disabled (as will be necessary during upgrades)
- how individual components should be started, stopped, and monitored
- what kinds of errors should be detected? what kinds of errors should be ignored, or escalated?
- what alarms should be generated when upset conditions occur?
- how should such alarms be delivered?
- how to save the current configuration setup
- how to find the current configuration setup (in an archive of backed-up setups)
- how to roll forward / roll back configurations
- when and how certain components ought to be disabled

The Replication Engine will handle computing the full set, sequence, and possible parallelism of what needs to be done at any particular time, and executing those operations.

Specifying those aspects would be done via a declarative configuration file rather than via imperative logic.[7] That should make it easier to extend and modify the configuration as the system evolves. Where it makes sense, an analytic engine could compute all the important relationships and required actions, and then call external scripting to perform individual actions which are specific to particular components.

Possibly, this configuration and execution could be managed by an existing tool such as cfengine. (http://www.cfengine.org/), the core of which is now licensed under GPLv3 (with commercial support also available). We would need a little investigation and experimentation to see how well such a tool would fit these needs.

If we instead implement our own tool for this, we see the following components needing development:

- a Replication Configuration File format
- Replication Configuration File content for the particular components we need replicated
- a Replication Engine to analyze the Configuration File, connect to other machines, and execute actions

C.3 Structure

[which files comprise the entire replication control engine and its data repositories]

C.4 Configuration

The following types of data will need to be part of the Replication Engine configuration:

- files and file trees to be replicated, and how they are to be grouped with respect to the applications that are affected by them
- replication schedules for various files and file trees
- (future) validation scripts to run before installing new files and file trees (to help protect against grabbing and installing inconsistent or incomplete configurations)
- what processes need to be quiesced or blocked for the duration of sets of replication activities

7. See http://en.wikipedia.org/wiki/Imperative_programming and http://en.wikipedia.org/wiki/Declarative_programming for explanations of these terms.

- local ports or scripts or other identifying information used to quiesce, block, or bounce applications
- (future) how to trigger other applications into re-reading their configurations
- (future) how to trigger other applications into reading the replication state and enter or exit read-only mode with respect to allowing changes to their own configurations to be made through those applications
- what data transformations need to be made to which files, and how they should be carried out
- scripts and programs to call for data transformations, for replication transport, for replicated file installation, etc.
- location of the pending/backup configuration repository
- location of the replication state file (for simple viewing of the current state by other applications; and for storage of the replication state across bounces of the Replication Engine)
- location of the server to which this one is paired
- port number of the remote paired replication server (or its DR heartbeat connection)
- time-related data:
  - DR heartbeat period
  - number of failed DR heartbeats before initial paired-server failure is recognized and Notification Authority is arrogated
  - some measure of DR heartbeat flapping that also drives paired-server failure recognition even when we never have enough consecutive failed heartbeats to recognize paired-server failure
  - number of consecutive successful DR heartbeats before paired-server up status is believed after failure and Notification Authority is relinquished
- the number of old configuration sets to preserve for each application, as replication rolls forward
- whether a heretofore uninitialized system is to operate as a Primary Server or as a DR Server
- where special commands such as enabling/disabing notifications or sending state-transition notifications or Replication Engine alarms must be sent
- where to probe Nagios state to ensure that it is consistent with what the Replication Engine believes
- Replication Engine log file path
- logrotate setup for all Replication Engine related files
- scripts to run to tell whether paired databases can be brought into sync by processing existing transaction logs, or whether wholesale truncation and re-population is needed

We need a general format for the configuration files, something like what Config::General or TypedConfig supports, or perhaps libconfig. And then in the design we need to document the specific directives that will be supported, and their syntax and relational constraints. Here's an example of a readable flexible format, though perhaps not matching any of the above:

```
zone "google.com" {
    type forward;
    forwarders {
        123.45.8.24;
        78.94.23.5;
    }
    forward only;
}
```

YAML (http://www.yaml.org/) seems to be the config file structure of choice when choosing a format that can be processed cross-language, as may be needed for saving replication state to be read by third-party applications to see if they should run in read-only mode with respect to changing their own configurations. A web posting recommends it for the following qualities:

- YAML documents are very readable by humans.
- YAML interacts well with scripting languages.
- YAML uses host languages' native data structures.
- YAML has a consistent information model.
- YAML enables stream-based processing.
- YAML is expressive and extensible.
- YAML is easy to implement.

Bindings for a wide variety of compiled and scripting languages already exist.

C.5 CLI Commands

This section is a reference manual for the commands accepted by the Replication Control Command-Line Interface (CLI). A CLI is chosen over a GUI partly because of the simplicity it affords for remote and scripted control, and partly because a GUI would be dependent on a supporting infrastructure for authentication/authorization and display. If we tried to use the local GroundWork Monitor system to provide that infrastructure, it might backfire because we might need to bounce that GroundWork Monitor in the course of executing commands. On the other hand, once we have a CLI built, it may be possible in a future product version for it to have a mode in which it acts as a CGI and can run in a portal window to provide status information and possibly some limited control, as long as the web infrastructure is operating. Or an extended version might provide its own specialized web server on a designated port, or we could provide a local Perl/Tk graphical interface.

Conceptual list of commands:

allow Notification Authority to be determined by current conditions

grab Notification Authority (and lock that choice)

release Notification Authority (and lock that choice)

grab Master Configuration Authority

release Master Configuration Authority

display system DR state (on both Primary and DR systems)

display DR heartbeat results

block sync operations (globally, or per-component)

unblock sync operations (globally, or per-component)

> [block and unblock operations are to control automated actions; they will not block manually-initiated synchronization operations]

initiate sync operations (in either direction, following which machine currently has Master Configuration Authority and which does not; with subcommands for individual databases and for sets of files related to particular applications)

[*See the pseudocode file for a more complete list, to be copied here and expanded, along with a description of interactive mode and command-line options.*]

Snapshot-of-work-in-progress list of commands:

```
help
login [<username> [<password>]]
engine start
engine stop
status all
status heartbeat
status [local|remote] engine
status [local|remote] notify
status [local|remote] config
status app <app-name>
status db <database-name>
notify allow
notify disallow
notify grab
notify release
notify lock
notify unlock
config grab
config release
pulse    # forcibly run a full heartbeat cycle if one is not in progress
block [sync] all
block [sync] app {all|<app-name> ...}
block [sync] db {all|<database-name> ...}
unblock [sync] all
```

```
unblock [sync] app {all|<app-name> ...}
unblock [sync] db {all|<database-name> ...}
diff all
diff app {all|<app-name> ...}
diff db {all|<database-name> ...}
sync all
sync app {all|<app-name> ...}
sync db {all|<database-name> ...}
commit app <app-name>      # only a single application, for now
commit db <database-name>  # only a single database, for now
list app <app-name>        # lists available backup timestamps for this app
list db <database-name>    # lists available backup timestamps for this db
rollback app <app-name> [timestamp]
rollback db <database-name> [timestamp]
quit
exit
```

C.6 More Stuff

- What kinds of queries might one want to make of the DR Engine? How should it, itself, be monitored?
- How should the configuration of the DR Engine itself be replicated? What transformations should be applied during such replication?
- Here is a suggested hierarchical directory structure for storing scripts, for actions taken globally, at a subsystem level, and within particular applications:

  `/etc/logrotate.d/groundwork-replication`
  : Log rotation specifications for all log files related to DR (see `/usr/local/groundwork/replication/logs/` below).

  `/usr/local/groundwork/replication/actions/`*category*`/`*object*`/`*scripts*
  `/usr/local/groundwork/replication/apps/`*my_app*`/actions/`*scripts*
  `/usr/local/groundwork/replication/apps/`*my_app*`/resources/`*my_type*`/actions/`*scripts*
  : Directories for scripts at the global, application, and intra-application level, such as for starting, stopping, and monitoring particular applications. These scripts would have standardized names, so their simple presence under those names would be sufficient to get them triggered at corresponding times, without explicit configuration. Scripts are named with a canonical *object*`.`*action* convention, with the standard actions being `capture`, `stop`, `deploy`, and `start`.

  `/usr/local/groundwork/replication/bin/`
  : All general scripts and compiled binaries which are not specifically related to automatically triggered DR transitions.

  `/usr/local/groundwork/replication/config/`
  : Configuration files for all DR scripts and programs.

  `/usr/local/groundwork/replication/perl/`
  : Perl packages needed for the Replication State Engine which are specific to the Engine functionality, or standard CPAN packages needed beyond those which are shipped with GroundWork Monitor, or more up-to-date versions than were shipped with the versions of GroundWork Monitor that might be used with the Replication State Engine.

  `/usr/local/groundwork/replication/var/`
  : Dynamic information files, such as current-state data.

- A core engine will need a variety of scripts to take actions in particular situations. The nature of these scripts includes:
  - running heartbeats
  - starting individual components (including all aspects of getting them ready to run, including verifying that they each have a complete and valid configuration in place and correcting or alarming on misconfigurations)
  - stopping individual components
  - monitoring individual components
  - capturing component state

    - restoring component state
    - transferring data between machines
    - alarm generation
    - configuration backup
    - configuration restore
    - failover sequencing and user exits
    - switchback sequencing and user exits
    - component error detection and recovery
    - component enabling and disabling
    - pre- and post- action scripts as well? or scripts for other conditions or transitions?
    - global action scripts; global pre- and post- action scripts as well
    - scripts at the global, subsystem, application, and component levels
    - scripts that ought to operate specifically on one server but not on both servers of a Primary/DR pair (say to condition certain site-specific communications equipment)
    - etc. etc.; see Components for further examples
- what generalized parameters will these various types of scripts take? (name of application; application instance tag; type of action; logging level; etc.)
- specified sequences of script invocations under various conditions

Appendix D: Design Requirements and Choices

Whatever we build as a DR prototype for the first customer, we know that a critical distinction must be made between on-line and off-line operation of the DR server while the Primary server is operating. For a product version, some customers will be happy with a hot (on-line) DR server, while other customers will insist that normal (Primary-based) operation should not invoke the extra client and network load of monitoring from the DR server as well, so an off-line DR server should be supported. Whether or not the DR server monitors the rest of the infrastructure, it will continuously monitor the status of the Primary server so it can tell when to assume responsibility for monitoring and sending out notifications.[8]

D.1 Requirements Resolution

For convenience in comparing the solution to the original requirements, the structure of this section mirrors that of Section 1.1, Customer Requirements. Where necessary to conform to the standard terminology listed in the Glossary, "Standby" in the text of the original Customer Change Order has been replaced by "DR" here.

D.1.1 DR Monitoring Coverage

The DR system must provide the functionality of the following subcomponents in the event of a Primary system failure.

- *Perform active Nagios polling of configured devices and services.*

  The DR server will generally act as a hot spare. In that capacity, it will perform active Nagios polling on an ongoing basis, even when not operating in DR mode.

  Some customers may wish to instead use a contingent (warm spare) DR server, which would not perform active monitoring until and unless the Primary system (or communication with it) has failed. Hence an eventual product solution should support a configuration switch to control the active or inactive state of polling from the DR server in Normal Mode operation.

- *Perform active Cacti polling of configured devices and services.*

  (on DR, replicated Cacti [how?]) (Cacti database replication, being mindful of possible active Cacti polling; locally configurable as a hot or warm spare)

- *Provide the same SNMP Trap processing capabilities as the Primary.*

  The DR server will be configured to provide SNMP trap processing. Since SNMP traps originate at the leaf devices and not from polling by the server, to take advantage of the DR setup, individual devices must be configured to direct their traps to the DR server as well as the Primary server. For devices which can only be configured with a single target address, this might be done by using a virtual IP address. The site must then either remap this virtual address through an automated or manual process during a DR event, or otherwise replicate the trap packets to both Primary and DR servers on an ongoing basis. Devices which are not configured to send traps to the DR server in one way or another will be invisible to the DR server.

  Scripts which process SNMP traps will be automatically replicated from the Master server to the Slave server.

  SNMP MIBs directory must be replicated, as well as snmptt conversion configuration files.

  Main repository: `/usr/local/groundwork/common/share/snmp/mibs/`

  Also look around here: `/usr/local/groundwork/common/etc/snmp/CISCO-C2900-MIB.conf`

  Also look in snmptt.ini and in the snmptt script itself, for possible other locations.

  Use the synchronization scripts from GW Connect. Take a look at them.

- *Provide the same SYSLOG message processing as the Primary.*

  The DR server will act as a hot spare with regard to syslog message processing, on an ongoing basis even when not operating in DR mode. (Replicate syslog-ng.conf, plus directory structure for log

8. For an off-line configuration, this means that almost all monitoring would be disabled until a failover event occurs. So we would need a simple mechanism to selectively enable and disable large portions of the monitoring.

files, plus plugins [e.g., a possibly locally-customized check_syslog_gw.pl script, plus a chocolate version of check_syslog_gw.pl from one of our Technical Bulletins].) [might we use spooling of any sort?]

Work out how the sending software will get messages to both copies of syslog-ng. Presumably, the sources will need to send to both Primary and DR servers. Those sources that are only configured to send to one server will be out of luck – how common will that situation be? A back-end mechanism will sync the config files. Take a closer look at how syslog-ng is configured, to see how its own configuration might interact with messages sent to both servers.

Use the synchronization scripts from GW Connect. Take a look at them.

- *The DR server must reflect the runtime states of the Primary for scheduled downtime, comments, and external commands.*

  The Status Viewer UI will submit these changes as Nagios commands to the Bronx command channel on the Primary server, and it (the Status Viewer) will be configured to also submit the same changes to the DR server. This will involve some changes to the Status Viewer so a failure to send to both servers is reflected to the UI, so the user becomes aware of the failure.

  GroundWork will not modify the Nagios CGIs so they similarly forward such changes to the Slave server. Therefore, the Status Viewer must be used to submit all such changes to the Master server.

  Scheduled downtime and comments are stored in the `nagios/var/status.log` file. The effects of external commands are stored in the `nagios/var/nagiosstatus.sav` file. To synchronize Master and Slave after a link outage or other failure during which one could not simply forward scheduled downtime, comments, and external commands to both Primary and DR systems, it may be best to just synchronize the states by grabbing the Master system `nagiosstatus.sav` file and installing it on the Slave system, then bouncing nagios on the Slave system.

  This area needs consultation with Roger, design, and testing.

D.1.2 DR Notification

In the event of primary server/system failure, the DR server/system must provide the same notifications as would have been sent by the Primary system.

Failover and switchback processes with respect to notifications will be automated. (We need documentation on the existing mechanism for this; see doc in PE Training and in Subversion training.) [expand this]

within the limits of Nagios (submit a Nagios command to enable/disable notifications, perhaps); see the monitor_parent.pl script for clues

intercept notifications, and either block or pass them; look at host-notification and service-notification commands; race conditions? what special conditions and actions arise during critical time windows around the timeframe of a link failure?

D.1.3 DR Operator Interface

The DR system must provide the same functionality as the Primary in the event of a Primary system failure. This functionality includes at a minimum the following interface elements:

- *Dashboards*
  Dashboard definitions and authorization data are both stored in the `jbossportal` database, which will be periodically replicated to the alternate server. A technical challenge is that the `jbossportal` database is managed by Hibernate, so the timing of updates to the database will be uncertain without careful formal handshaking with Hibernate. (The EHCache layer can possibly be used to replicate data across multiple Hibernate instances, though we would need to test performance of large updates [would they be slowed down even on the source machine, let alone what might happen on the target machine due to high link latency?].) To obtain consistent and useable data on the Slave Server, we need some kind of atomic replication, meaning that we need to either flush all pending Hibernate updates and quiesce Hibernate during replication, or provide some kind of per-transaction replication capability.

[ask Roger about details of possible interactions with the cache, and in particular how to force Hibernate to flush everything to the database while (temporarily) not accepting any further change requests]

We need to replicate the entire `jbossportal` database. It would be easier/better to lock and unlock all tables around the replication, rather than bouncing the applications that use the database. However, that raises issues of what order in which tables should be locked, if we cannot get Hibernate to flush all its pending changes and know that everything in the database is complete and consistent. If we did have to bounce applications, that could be done by bouncing gwservices.

- *My GroundWork (Dashboards)*
  Dashboard definitions and authorization data are both stored in the `jbossportal` database, which will be replicated to the alternate server.
- *Event Console*
  We need to save filters (both public and custom), any action scripts added by the site, and the definitions of scripts and consolidation rules.

  Configuration files associated with the Event Console include:

  ```
  /usr/local/groundwork/config/console-admin-config.xml
  /usr/local/groundwork/config/console.properties
  ```

  The following database tables are used to store configuration data related to the Event Console *[this list needs review by Arul and Roger]*:

  - GWCollageDB.Action
  - GWCollageDB.ActionParameter
  - GWCollageDB.ActionProperty
  - GWCollageDB.ActionType
  - GWCollageDB.ApplicationAction
  - GWCollageDB.ApplicationType
  - GWCollageDB.ConsolidationCriteria
  - GWCollageDB.OperationStatus

  [use a lock on the read side when these tables are captured; that's an option (--lock-tables) on the mysqldump command line; or try the –single-transaction option instead]

  [look at Bookshelf for config info on Event Console]

  [In a warm-DR setup, we would want to daily replicate the last day of GWCollageDB.LogMessage and perhaps GWCollageDB.LogMessageProperty content, keeping careful track of the timespan-endpoint of each pass of such replication so we don't have either overlap or missing data in the replication. Such updates could actually be run as often as desired to keep the Primary and DR systems from getting too far out of sync (say, once per hour), as long as we keep track of the interval endpoint as noted, to start picking up new data from there.]
- *Status Viewer*
  Local configuration for Status Viewer is found in the `/usr/local/groundwork/config/status-viewer.properties` file. This file will be extended to support DR by including settings for forwarding commands and such to a remote secondary server.

  With a standalone server, there are typically no local configuration changes for Status Viewer beyond the settings that ship with the product, unless you have separated Foundation from running on the same machine as GroundWork Monitor, in which case there would be some adjustments made to communicate with the remote Foundation. Similarly, as we will make changes to Status Viewer to submit user scheduled downtime, comments, and external comments to Nagios, there will be a need for a small local configuration store to point to the remote Bronx on the secondary server where these Nagios commands should also be sent. This configuration might not be replicated between Primary and DR servers, as it will largely just be each pointing to the other and would require transformation to mirror that configuration when replicated. (Explicit server names will replace the use of “localhost”, at least for the secondary sink on each server.)
- *Reports*
  We need to replicate custom BIRT reports, performance view definitions, and log reports.

[getting the log reports back into the product so we support them in DR is a non-trivial task. it's not a bug issue, it's a license issue with certain widgets.]

[need details; talk to Thomas]

log reports: /usr/local/groundwork/gwreports/...

- *Nagios*
  We need to replicate any locally-provided and locally-modified plugins, as well as any custom CGI scripts installed by the customer.

```
/usr/local/groundwork/nagios/libexec/...  (plugins)
/usr/local/groundwork/nagios/sbin/...     (CGIs)
/usr/local/groundwork/nagios/share/...    (images, stylesheets, and similar
stuff that might be needed by CGIs)
```

- *NMS:*
  - *Cacti*
    (is everything contained in the Cacti database? no, there are templates (where stored? ask Dr. Dave), and perhaps other stuff – look under the installed cacti root directory)

    Cacti database
    `cacti` database (single instance, which is pointed to from what fixed location? where are the access credentials stored?)

    resource file tree
    `/usr/local/groundwork/nms/applications/cacti/resource/...`

    scripts
    `/usr/local/groundwork/nms/applications/cacti/scripts/...`

    plugins
    `/usr/local/groundwork/nms/applications/cacti/plugins/...`
  - *NeDi*
    We need to replicate the NeDi database and master configuration files.

    NeDi script
    `/usr/local/groundwork/nms/applications/nedi/nedi.pl` (sometimes needs local editing, and thus needs replication)

    NeDi database
    `nedi` database (single instance, which is pointed to from what fixed location? where are the access credentials stored?)

    NeDi master configuration file
    `/usr/local/groundwork/nms/applications/nedi/nedi.conf`

    NeDi system objects
    `/usr/local/groundwork/nms/applications/nedi/sysobj/...`

    NeDi cron jobs

```
0 4,8,12,16,20 * * *
(/usr/local/groundwork/nms/tools/perl/bin/perl
/usr/local/groundwork/nms/applications/nedi/nedi.pl -clo ;
/usr/local/groundwork/nms/tools/automation/scripts/extract_nedi.pl )
> /dev/null 2>&1
```

```
0 0 * * * /usr/local/groundwork/nms/tools/perl/bin/perl
/usr/local/groundwork/nms/applications/nedi/nedi.pl -clob >
/dev/null 2>&1
```

  - *Weathermap*
    We need to replicate maps. The rest of the configuration data resides within the Cacti configuration. (plus image files referred to by config files)

    everything related to weathermap
    `/usr/local/groundwork/nms/applications/cacti/plugins/weathermap/...`
    and not the copy in `/usr/local/groundwork/nms/applications/weathermap/...`

exclusions (files not to be syncronized)
`/usr/local/groundwork/nms/applications/cacti/plugins/weathermap/output/...`

maps
`/usr/local/groundwork/nms/applications/cacti/plugins/weathermap/configs/...`

custom images and images referred to by config files
`/usr/local/groundwork/nms/applications/cacti/plugins/weathermap/images/...`

- *ntop*
  We need to replicate the configuration file, plus any localized plugins. (ntop needs to be quiesced during replication.)

  [*How do we handle conflicting copies of ntop and its setup from the base product and from NMS?*]

  plugins
  `/usr/local/groundwork/nms/applications/ntop/lib/ntop/plugins/...`

  `/usr/local/groundwork/nms/applications/ntop/lib/plugins/...`

  configuration file
  ntop.conf (?) (send email to Dr. Dave to ask where this is located)

  Compare:

  `/usr/local/groundwork/common/etc/ntop/...`

  `/usr/local/groundwork/nms/applications/ntop/etc/ntop/...`

  to see which copy is actually used. (One may be from the base product, and one from NMS.)

  databases
  `ntop` uses `gdbm` for its internal databases. Unlike `dbm` files and `ndbm` files, the `gdbm` database files are not sparse, meaning there is no concern about them expanding during an ordinary copying process.

  `/usr/local/groundwork/nms/applications/ntop/db/...` (may be running output and therefore excluded from replication)

D.1.4 DR Administrator Interface

The DR system must provide the following administrator user interface elements in the event of a Primary system failure:[9]

- *My GroundWork (configuration)*
  Dashboard configuration data is stored in the `jbossportal` database, which will be replicated to the alternate server. In the absence of database replication, gwservices can be stopped to quiesce this database in order to synchronize it safely.
- *Auto Discovery*
  Several disk files used to store such things as discovery schemas, automation schemas, and perhaps other kinds of templates must be replicated.
- *Administration*
  Administration data for Foundation (and access credentials?) is stored in the `jbossportal` database, which will be replicated to the alternate server. In addition, we need to replicate the login-config.xml file, after identifying which of three copies in the distribution is the one that matters. (That file is used for LDAP support; we will need to bounce gwservices to load a new copy, and to start LDAP.)

9. Use of Standby Administrator user interface elements implies that the system has assumed the role of Primary and has become the authoritative source for configuration information. To prevent configuration change collisions, the system operator must take measures to ensure that Primary to Standby replication is disabled. Replication of modified configuration information on the Standby to the recovered Primary system is a manual process that needs to be performed by the system operator. GroundWork will provide instructions on how this is achieved.

General configuration data for GroundWork Monitor is stored in this file tree:

```
/usr/local/groundwork/config/...
```

in addition to particular files there which are called out separately elsewhere in this chapter.

Also in this category, we need to replicate the `monarch` database, as well as any locally-installed profile files. Access to Monarch needs to be locked out on the Slave Server, perhaps through the JBoss portal administration.

- `monarch` database
- Monarch profiles

```
/usr/local/groundwork/core/profiles/...
```

- *NMS*
  - *Cacti*
    (ensure that access credentials are replicated; need to disable administrative components for slave operation)
  - *NeDi*
    (should not need anything extra for administration beyond the NeDi database and config files, etc. that are needed for user operation) (there are some admin features in NeDi; some database data, ability to change passwords; such stuff ought to be disabled in slave mode)
  - *Weathermap*
    (should not need anything beyond user-operation setup) (the Weathermap editor ought to be disabled in slave mode; look for control as a page portlet)
  - *ntop*
    (need to replicate the config database) (ntop has some admin menus, plus some admin plugins that ought to be disabled in slave mode)

D.1.5 General

The proposed modifications are intended to operate on Primary and DR systems in separate data centers over high latency links.

(high latency inveighs against solutions which might attempt to synchronously replicate all changes as soon as they are applied to the Master system)

Securing communications between the Primary and the DR server is possible via a number of commercially available methods and is beyond the scope of this document or the proposed modifications. Those protocols are as follows:

- *Database replication via MySQL socket communication* — *TCP/3306*
- *Nagios Service Check Acceptor (run-time updates for acknowledgments and downtime scheduling, etc.) protocol with DES encryption* — *TCP/5677*
- *MySQL snapshot replication via SSH* — *TCP/22*
- *Flat file replication via SSH* — *TCP/22*
- *Control signals via SSH* — *TCP/22*
- *Heartbeat signals via unencrypted TCP socket communication* — *TCP/5699*

The DR system will assume all described functions of the Primary system within 10 minutes of detecting a Primary system failure. A network split or any other communication problem between the Primary and DR is defined as a Primary system failure in addition to other known failure modes.

(Heartbeat signaling is currently covered by the monitor_parent.pl script; take a look at it.)

(for a health check, check all the services we start in our various ctl.sh scripts)

(our initial assumption is that a once-per-day replication is assumed to provide sufficient mirroring, rather than continuous on-the-fly copying; continuous replication would be the province of an HA solution rather than a DR solution, so the DR solution admits a window of possible loss of configuration data, though the replication can be manually triggered outside of the regularly scheduled copying to limit this window)

D.2 Technical Requirements

Certain capabilities are needed to support the stated Customer Requirements.

xxx

D.2.1 Supported Configurations

The first customer's configuration involves only a Primary server, with no associated Child servers. The NMS add-on will run on the Primary server.

In a more general solution, addressing the needs of other customers as well, these configurations represent the principal system which will be mirrored by the DR system.

- GroundWork Monitor Primary server
- GroundWork Monitor Primary server, plus NMS on the Primary server (*the setup of the first customer*)
- GroundWork Monitor Primary server, plus NMS on a separate server
- GroundWork Monitor Primary server (Parent), plus Child server(s)
- GroundWork Monitor Primary server (Parent), plus Child server(s), plus NMS on the Primary server
- GroundWork Monitor Primary server (Parent), plus Child server(s), plus NMS on a separate server

Not considered at this time:

- support for n+1 Child monitoring servers

D.2.2 General Requirements

xxx

D.2.2.1 Review of Configuration

Preceding sections have mentioned particular files and file trees that will need replication and possible transformation. In addition, GroundWork will need to review the broader server configuration, to see what other data repositories need attention. For instance, most of the `*.properties` and `*.xml` files in the `/usr/local/groundwork/config/` directory deserve some investigation to see whether they should be subject to replication.

D.2.2.2 Extra Hardware Requirements

If we use standard MySQL replication facilities for keeping Primary and DR databases synchronized, that mechanism depends on binary logging on the Master server. And there is no way to restrict such logging to only the relatively static small set of configuration data; even the high-volume dynamic data changes will be logged. This means a tremendous extra load of disk i/o, and if the binary log is on the same spindle as the database itself, this can result in “head rattle”, where even more time is lost due to continually switching back and forth between areas of the disk. Such a setup can seriously degrade performance. We may therefore need to require that a DR setup use a second spindle for database logging, especially for moderate-to-large customer configurations.

D.2.2.3 DR Server Configurable as Hot or Warm Spare

The DR prototype will treat the DR system as a hot spare in all regards. But as a general product feature, some customers may wish to avoid the extra ongoing monitored-system and network loading imposed by an active DR server while the Primary server is operating normally. Whichever type of spare is supported, several technical issues arise:

- mechanisms for safely replicating data, whether or not the backup server is currently active
- operational processing delays imposed when activating a warm spare or deactivating a hot spare
- possible data duplication or loss when a backup server is transitioning into or out of hot-spare mode

D.2.2.4 Data Integrity

Methods which synchronize data between Primary and DR servers must take into account the fact that the Primary copy of individual pools of data may be subject to modification at any time. This could mean that the data is in a state of partial change at the time the synchronization method comes around to work

its magic. It is therefore necessary that the synchronization mechanism be able to detect incomplete sets of changes, so it can avoid propagating an inconsistent or broken configuration to the other Server.

xxx

D.2.3 On-Site Disaster Recovery Testing

A serious installation of a DR setup has to include testing the DR capabilities well ahead of an actual unanticipated failure event. Such testing must involve running through the documented procedures for failover and switchback operations, to ensure that:

- the procedures actually work;
- the staff is familiar with the formal processes and where to find detailed instructions in an emergency;
- any supporting requirements are actually in place.

To support the site in such testing, GroundWork must provide documentation listing the processes to follow in possible failure and recovery scenarios, along with resources that must be provided for the total solution to work (e.g., particular ports be open through firewalls).

D.2.4 Notification Authority

Talk about:

- *the assumption of Notification Authority, including delays and hysteresis in recognizing that notifications must be sent and that Notification Authority must be presumed or dropped*
- *double notifications, from both Primary and DR server, in certain situations, due to race conditions in the presumption and dropping of Notification Authority and the desire never to miss notifications*
- *notifications must be tagged with their source, so the receiver can tell whether they originate from the Primary Server or the DR server*

D.2.5 Master Configuration Authority

Unlike Notification Authority, transfer of Master Configuration Authority must be a manually-initiated process. This constraint is present partly to prevent serious confusion that would arise with an intermittent connection between Primary and DR servers, when Master Configuration Authority might be transferred back and forth between Primary and DR servers while an administrator is in the middle of making changes on one server or the other. It also ensures that the administrators become aware of which server they need to be accessing in order to make configuration changes; if human procedures must be in place to transfer this authority, then there can be a formal place in such procedures for the administrators to be told which server is currently considered to be the Master.

Delegation or arrogation of Master Configuration Authority cannot be automated without more information than is available to an automated process. For instance, if the Primary and DR systems lose contact, that might be due to a severed link, a DR system failure, or a Primary system failure. The correct response in each of these situations is a bit different.

- For a downed link with both Primary and DR systems remaining up, Master Configuration Authority stays with the Primary system.
- For a DR system failure, Master Configuration Authority stays with the Primary system.
- For a Primary system failure, Master Configuration Authority will be transferred to the DR system, but only if the outage is expected to last for some time.

Similarly, in the reverse direction, Master Configuration Authority may not be immediately transferred back from the DR system to the Primary system immediately upon restoration of the Primary system. For instance, if the site has really undergone a serious malfunction at the primary site, staff there may be too busy restoring the rest of the infrastructure to take on the duties of managing the monitoring configuration, for some time after the Primary server itself is back up.

Whichever system actually has Master Configuration Authority must be very visible to the administrators working on both the Primary and DR servers, so no confusion can arise as to changes accidentally made on the wrong server. Where possible, changes should be automatically blocked on a Slave server, and

a notice of whether the server has Master Configuration Authority should be clearly visible in many if not all of the configuration screens.

D.2.6 Installation Procedures

Basically, on each side:

- Install RPM.
- Configure.
- Start.
- Enable dynamic notifications on both sides.
- Grab Master Configuration Authority on Primary, release it on DR.

The actual installation process is much more detailed. See the separate installation document.

D.2.7 License Integration

Full DR support represents a new capability, and as such it should be subject to license restrictions, with a new Order SKU created for this product. Given that the DR system monitors the same systems as the Primary system, the actual license key can and should be shared between the Primary and DR systems, and the license key file will be subject to replication like other configuration data.

D.2.8 Performance

(extra i/o time and disk-head occupancy needed for binary logging on master database, to support replication; it may be necessary to require a second spindle for the binary log if this load gets heavy, so as not to slow down normal operation of the server)[10]

(max down time during successful sync operations)

D.3 Failure Scenarios

xxx

- Parent down, Child still up; use Child for monitoring, to feed to DR

xxx

D.4 Operational Sequences

We need to consider all possible situations where the Primary and DR servers will be up, down, or in transition.[11] This must cover both graceful and sudden events.

D.4.1 Primary Server Startup

There may be a window of uncertainty at startup as to whether a Primary or DR server has master authority. We might want some kind of hold-down timer implemented until the situation can be fully detected.

Startup activities must be integrated with synchronization so that if a bad or partial synchronization is detected during startup, the system will automatically roll back to a safe configuration before startup.

D.4.2 DR Server Startup

xxx

10. Once MySQL replication is turned on, all changes to the master database are logged, not just the ones the slaves are interested in. That means that the high-volume updates to the database, which will in general not be replicated, will nonetheless be logged. This will induce extra load on the master to record this data, and extra load on the slave to slog through it looking for a few gold nuggets.

11. Not to do so would be like writing a loop in a programming language and forgetting about initialization and/or termination conditions.

D.4.3 Communication Failure During Synchronization

If the communication link can fail during ordinary monitoring, then it follows that it can also fail while synchronization operations are ongoing. Recognition of this possibility imposes certain architectural requirements on how synchronization is monitored and controlled.

- At the time of a communication failure, parts of either the Primary or DR system might be intentionally down. Every effort must be made in such a situation to automatically restore a working configuration to each side and to bring them both up again. In general, that suggests that most changes should be first staged, then brought into play with a few quick atomic file renames or similarly operations when possible. Such actions would be fast, easy, and reliably reversed.
- The link status must be monitored independently on both the Primary and DR systems, so that each side can recognize the failure and restore its own condition without assistance from the other side.
- The link status must be monitored outside of the Primary and DR monitoring systems, because a link failure must be detected while synchronization is occurring with these systems temporarily down.
- All synchronization operations must be defined with a well-defined process for rollback to safe sequence points in case full sets of sync operations cannot be completed due to link failure.
- Link failure handling must include some kind of hysteresis so as not to suffer from serious flapping while synchronization operations are ongoing. For administrator convenience, it would be good to create a graph of link availability, both raw and filtered, with data collected independent of the monitoring system (and while the monitoring system itself is down).
- Synchronization operations must have a locus of control on each end of the connection. For instance, one cannot simply ssh over from the Sync Master to the Sync Slave to carry out certain operations, as that would leave the Slave stranded in case of a link failure. Instead, the Sync Master must send commands to the Sync Slave, and the Sync Slave must be capable of carrying out those actions on its own, reporting back status and initiating rollback and bring-up actions if the link fails before all sync operations are done.
- Some sort of two-phase commit transactions (handshaking) between Sync Master and Sync Slave controllers must be used to send commands and return execution status. The explicit acknowledgments in such a protocol are necessary for each side to know whether the other side has received the notifications sent.

D.4.4 Start of Communication Failure During Monitoring

Probing between Primary and DR systems must detect not only “is the link up”, but also “is monitoring up on the other end of the link”.

The actions described here happen on both the Primary and DR systems. When a peer goes down:

- turn on active polling if the system was previously inactive [a Nagios command will do this]
- turn on notifications
- assume master notification authority
- send a notification that the link is broken

In part, this means that notifications will be sent from both sides.

D.4.5 End of Communication Failure During Monitoring

When the link returns to full operation:

- continue to probe periodically; wait until it appears to be stable (according to some algorithm involving hysteresis)
- transfer Notification Authority to the Primary server, but only if Master Configuration Authority is already enabled there

D.4.6 Primary Server Downed

This situation is specific to a condition where the Primary server can be contacted from the DR server, so the link is known to be up, but the Primary monitoring is found to be not operating adequately. All action desribed here therefore takes place on the DR server.

- turn on active polling on DR
- turn on notifications (assume Notification Authority)

D.4.7 Primary Server Restored

xxx

D.4.8 DR Server Downed

xxx

D.4.9 DR Server Restored

xxx

D.4.10 Primary Server Shutdown

(fail over naturally during scheduled Primary server downtime, or bring up DR server beforehand)

D.4.11 DR Server Shutdown

xxx

D.5 Operational Transitions

xxx

D.5.1 Configuration Updates

xxx

D.5.2 Failover

xxx

D.5.3 Failback

xxx

D.5.4 Switchover

xxx

D.5.5 Switchback

xxx

D.6 Future Features

xxx

D.6.1 Additional Add-on Packages

- NagVis: will need synchronization of database entries, configuration settings, and glyphs
- fping configuration

D.6.2 Special Integrations

- Remedy ticketing system
  BMC Magic ticketing system

Appendix E: Technical Issues and Design Alternatives

xxx

E.1 Major Technical Issues

xxx

E.1.1 Principles

xxx

- fail-safe operation; passive safety features where possible, active safety features where necessary

E.1.2 Design Issues

xxx

- identify the full set of configurations to be supported, both initially and over time for other customers
- identify all objects that need to be replicated to the DR server (individual databases or tables; individual flat files or collections of files)
- identify how not just monitoring data, but also performance data, will be handled between Primary and DR servers
- identify how often data must be synchronized between Primary and DR servers, for various categories of data (e.g., user info, monitoring configuration, user-specified console filters, states of external commands, etc.)
- identify how precisely the peer states and performance data will be precisely synchronized in time and space
- identify how replicated objects will be synchronized
- identify how the communication link and peer health will be probed
- identify how Master Configuration Authority will be saved, displayed, and used to control server actions
- identify how individual components (e.g., syslog-ng) and their own data sources must be configured to operate in this environment
- identify how the operational states of individual components will be transitioned during soft and hard DR events (e.g., turning on active polling on a warm DR server)
- identify all possible operational sequences of Primary/DR state transitions, and what will happen in each case
- identify performance goals (e.g., maximum time to replicate certain databases; maximum time for intentional outages during replication activities)
- identify which system components must become aware of whether the system is operating with Master Configuration Authority, and how they must behave if not
- identify how individual system components must synchronize their changes of configuration state with possible asynchronously-initiated replication activity
- identify how dynamically-changing data (such as updates to scheduled downtime) can be synchronized between servers
- identify race conditions in synchronizing servers, and what can be done to mitigate their effects
- identify the capacity, throughput, and latency, and bandwidth requirements of all the desired data replications, plus the recommended transports for these data transfers (e.g., NFS for the remote slave database reading the binary log on a master database)
- identify whether on-line or off-line replication is appropriate, and what happens to pending data changes in an on-line replication scenario during an outage
- identify security constraints affecting any proposed solution (e.g., the need for encryption; password storage; etc.)

- identify component dependencies (e.g., if we someday want replication control to be accessible via a GUI, then we cannot have it use the same Apache httpd server as the GroundWork Monitor instances it might need to bounce)
- identify what variant options ought to be supported (e.g., on-line vs. mostly-off-line DR server in normal-mode operation)
- figure out how to avoid common-mode failures that might occur if a bad configuration setup is installed on the Primary server and then immediately reflected to the DR server; ought there to be some configurable intentionally-imposed delay before configuration changes are propagated from Master to Slave? or safe backup points that can be rolled back to?
- to what extent should configuration changes be automated when transferring a configuration between Master and Slave? for instance, the monitoring-server hostname may be embedded within the configuration; monitoring-dependency chains may well change from the perspective of the different monitoring servers (e.g., different child servers may be used, and different network gear may stand between the monitored nodes and the different monitoring servers); different notification targets and different out-of-band notification mechanisms may be desired to raise the alarm with remote staff or to avoid common-mode failures
- If a site is truly in DR mode, they may have switched to backup application servers (say, at a wholly different location). So there may be additional systematic substitutions that ought to be made to the DR configuration when it is copied from the Primary system, or undone when the Primary server is updated from the DR system when it has Master Configuration Authority.
- how will GDMA on clients be configured to automatically redirect where the client picks up its configuration settings during a DR scenario?
- do we currently have an easy way to set up Child and Windows Proxy servers to send duplicate data streams to Primary and Standby/DR servers, for monitoring data, for logging streams, etc.?
- Will we use “localhost” or the actual machine name to probe self-health, and how does that affect replication of the configuration from Master to Slave? Is there any possible confusion, such as localhost meaning different things on the Primary and DR servers when applied to saved RRD file data? Also, might the nature of monitoring-server health-check commands be different (e.g., local vs. remote access to run commands) when viewed from the Primary and DR servers?

E.1.3 Implementation Issues

xxx

- identify how to make individual components aware of whether the system is operating with Master Configuration Authority
- make individual components pay attention to whether the system is operating with Master Configuration Authority (i.e., have those components operate in a read-only mode if the system does not currently have MCA)[12]
- identify how to make the local administrator aware of whether the system is operating with Master Configuration Authority

E.1.4 Documentation Issues

xxx

- identify which manuals must be upgraded with descriptions of capabilities and procedures

E.1.5 Testing Issues

xxx

- identify how to test each possible DR-event scenario (condition and transition)

12. An initial implementation might ignore trying to enhance various components to disallow changes without Master Configuration Authority. Given that MCA transfer is a manual process, it would be up to the site to advise administrators where changes are to be made, with the understanding that violations would be subject to automatic destruction during the next Sync operation.

E.2 Existing Scripting

Describe here the scripting we use for Standby servers; to what extent it might apply in the DR scenario; benefits and deficiencies of that scripting.

The following scripts are currently in use to support Standby servers.

heartbeat.pl
Calculates the health of the local server and notifies its peers.

ischanged.pl
Checks a file to see if it has changed since the last time this script ran. *FIX THIS: where is this invoked?*

monitor_parent.pl
Checks a distant server to see if it is running Nagios. Enables or disables notifications on the local server based on the result. Previous versions would also enable or disable active and passive host and service chceks on the local server based on the result, presumably to implement a warm spare instead of a hot spare. However, running Nagios is just one component of a full GroundWork Monitor system, so perhaps a more-thorough health check is in order.

MonarchFile.pm.diff
This is a patch to the base GroundWork Monitor release, that invokes the standby_group_update.pl script at certain times. The existing patch appears to be targeted at GW version 5.2.1.7 (only), based on the line numbering found in the patch. (I also have a contradictory suggestion that it's no longer needed, its functionality having been already folded into the GW5.2.1.7 release.) On the other hand, the same kind of patch has been applied at a customer site using GW5.3.

As an aside, the form of this patch needs simple but serious work to provide adequate context for inserting the new code at the correct place in the MonarchFile.pm file, as that file evolves and the line numbers change over time. As near as I can tell, it looks like the new code is supposed to go here within the build_files routine:

```
read_db();

# Professional Services callout to generate a host group that
# contains hosts that are not assigned to any monarch sub groups
# of the monarch standby group.  The monarch standby group name
# and the name of the new host group are presently hardcoded into
# the script that is called.
my $result =
    `/usr/local/groundwork/common/bin/standby_group_update.pl`;

if ($options{'group'}) {
        # processing a single instance
        process_group();
} else {
        # No group specified, so process standalone
        process_standalone();
}
```

install_standby.pl
xxx

standby_group_update.pl
xxx

E.3 Notes

This section is a holding ground for thoughts and ideas that should migrate elsewhere, or that I just want to think about further.

E.3.1 Unsubstantiated Ideas

- We need communication and alarms in both directions.

- During a failover, the latest changes made on the Primary system may not be reflected on the DR system, or may be partially reflected (and might possibly not represent a full, consistent configuration). During a switchback, those changes can either be ignored (lost) when the DR configuration is replicated back to the Primary system, or some attempt may be made to merge those original changes on the Primary server with subsequent changes on the DR server. A trivial implementation will ignore this problem and simply run with the last synchronized configuration, as subsequently updated on the DR system. The lost changes will then need to be manually re-applied if they were not subsequently made on the DR system before the switchback.
- The first step in a switchback operation must be to re-synchronize Primary and DR, with any changes made on the DR system while it had Master Configuration Authority reflected back to the Primary system. Only then can the Primary system be brought back into operation, and Master Configuration Authority transferred to it.
- Failover and failback of notifications must be automatic, while switchover and switchback of Master Configuration Authority must be manual.
- There must be some way for the system to display to the administrator whether it is in Master or Slave mode with respect to configuration changes.
- There is currently no mechanism available for Cacti replication.
- login-config.xml needs replication. But which copy? Should we just replicate the entire `/usr/local/groundwork/config/...` file tree and be done with it?

  ```
  /usr/local/groundwork/config/jboss/login-config.xml
  /usr/local/groundwork/config/login-config.xml
  ```

- Our current replication of Monarch/Nagios data uses “build instance” on the Primary server to push this data to the Standby. Or possibly Build and Deploy. This is a manual process. And we need to understand whether this really only transfers Nagios-visible constructions and whether associations that are completely managed by Monarch are not reflected to the Standby server's Monarch database.
- Read up on MySQL replication.
- What to do with RRD files? If the DR system is normally off-line, they will be mostly empty on the DR system. Replication of the RRD files might be considered unacceptably expensive.
- Full set of scripts to look at:
  - monitor_parent.pl
- If the Child servers are themselves mirrored at the DR site, then we may need to adjust their configurations as well, instead of just copying the Primary-site Child server configurations. Also, if both Primary Child and DR Child servers are on-line, what happens if we get duplicate results for a given monitored node, one from the Primary site and one from the DR site? Might those results be different because they follow different monitoring paths? Might that cause flapping?
- What extra cross-monitoring scripts and configuration files are installed by our Standby installation procedure, and how does their configuration differ between Primary and DR sites?
- How to safely upgrade a Primary/DR system, when the two servers are intentionally incompatible for a short period? How do we turn off synchronization for that period?
- How to initially install and then initially enable a DR system, with all its specialized capabilities?
- Maintenance changes on the Master machine may take place in stages, and not all the configuration parts may be ready for production just because they appear in a given configuration file or database. How do we prevent such in-transit changes from taking effect on the DR system until they are fully committed on the Primary system? (“Pending” vs. “In-Use” copies)
- Are there any issues with scripts that operate with internally cached state and status deltas, if they miss updates, that might cause differences in opinion between the Primary and DR servers as to the current state of a given node? Do we need some kind of periodic global status refresh to correct any accumulated divergence between the map and the territory?
- What does the 10-minute Primary-to-DR failover time imply about requirements for notification setup (e.g., intervals, retries) and calculations?

- phrase: “DR Parent with DR Child Servers”, for a full-up mirroring of the monitoring infrastructure; Primary Child sends to Primary (only), DR Child sends to DR (only); involves more extensive automated transformation of the replicated configuration; both Parent servers will monitor all Child servers, so no transformation is needed in that regard
- If the site used a redundant active WMI Child Server, then the respective Parent server configuration to send requests only to its own Child server would be another aspect of the configuration requiring transformation during replication.
- monitor_parent.pl: enabling/disabling notifications: is there hysteresis in this action? Compare to the time delay until notifications appear from within Nagios. You might have a possible complex interaction if the link is flapping, or if notifications that might have gone out don't because notifications are momentarily disabled; check to see how this affects processing of configured escalation timing
- To what extent is the set of nodes monitored by the DR server automatically maintained as a copy of what is monitored by the Primary server? That is to say, if we use a Monarch Group for this, how is that Group membership automatically updated as the set of nodes monitored by the Primary server changes? If we automate this, how might we allow for some nodes to be monitored only by the Primary server and not by the DR server, if a customer might need that sort of thing? Or would we just define that sort of thing out of existence?
- Do we want to depend on manually-initiated configuration pushes from Primary to DR system, as we currently do with Standby? Or keep them synchronized but only saved as Pending until we get manual confirmation or a daily automated full-synchronization step runs?
- Future versions of NMS may support multiple instances of Cacti. How will this affect DR setup?
- What happens if you want to upgrade just one component, say the version of Cacti running? How do you coordinate the replication actions to stop replication of just that component while the upgrade is taking place, before both Primary and DR servers are once again running the same version?
- The set of checks to be executed by a monitoring server may depend on the operating mode (normal vs. failover), along with which end of a Primary/DR combination is currently acting as Primary or Backup.
- We should enable a read-only mode of configuration viewing, in each monitoring component. This includes third-party products such as NMS components. We could perhaps give back code to other projects that would implement such a feature.
- Model-Driven Architecture (http://en.wikipedia.org/wiki/Model-driven_architecture)
- DASL: Basic object constraints and behavior are declarative, while additional object behaviors are specified procedurally as methods. Queries can be defined either declaratively or by writing methods.
- See also:
  - Linux-HA (http://www.linux-ha.org/)
- Some open-source Linux replication solutions:
  - rdist (http://www.magnicomp.com/rdist/ [the current (6.1.5) version already ships with Linux])
  - rsync (http://samba.org/rsync/)
  - DRBD (http://www.drbd.org/)

  See also:
  - Slony-I (for use with PostgreSQL; http://www.slony.info/)
- Some transforms seen before, even not in a DR configuration:
  - Certain checks are passive on parent vs. active on child, and need to be automatically transformed as such during export to the child. How is this done?
  - Host health checks for a child server may be on in the parent but off in the child; this setting must be automatically transformed during export to the child. How is this done?
- What call-out mechanisms do we currently support for transforms during configuration export?

- GUI issue: we cannot stop/start the local GroundWork Monitor from the GUI because it is intimately tied up with gwservices / login authentication and authorization; so we need CLI tools for control and status.
- For the prototype, use database replication of databases such as cacti and monarch.
- In maintenance mode, when we shut down apache httpd, there should be some way to redirect users to a "now in maintenance mode" screen. But how can this be done if httpd itself is down?
- We need a way to manually initiate synchronization, presumably through the CLI interface.
- For replicating configuration, does it make sense to deal with file timestamps and hashes, and only try to send deltas rather than full replacements? This might be done on a file-by-file basis, specified in the replication control configuration. rsync seems to have facilities for this sort of thing already implemented.

E.3.2 Open Questions

These questions are meant to provoke careful thinking.

- What conditions should constitute a failure that will invoke failover processing?
  - How will those conditions be detected?
- How will the Primary and DR systems communicate to determine whether Master Configuration Authority should be transferred in either direction? Answer: it will never be transferred automatically. But the DR heartbeat will communicate states, so each side will know whether the other side currently has Master Configuration Authority.
- How will client machines be told to redirect their active sending of results to the Primary and DR systems? In particular, this must be addressed for SNMP Traps, and for syslog messages. Answer: the devices must be set up in advance, as part of the installation process, to send traps and syslog messages to both targets. We will need to make this a checklist step for the overall DR package installation. And we need to test this design and what happens when traps and syslog messages cannot be sent because the target is inaccessible.
- To what extent will not just configuration data be synchronized between Primary and DR systems, but also operational Comments, Downtime Scheduling, External Commands, and so forth? Answer: all of those things need to be synchronized. In normal operation, the Status Viewer will send to Bronx on both targets. Recovery from failure mode must take into account the possibility that some such messages may not have gotten through, and perform some kind of wholesale re-sync to ensure matching setups. It's also possible that some such messages might be dropped even before failure mode is recognized, so a periodic Nagios state comparison should be performed to see if the two systems are actually still synchronized, and to repair any mismatches if not. In the alternative, we could have Status Viewer create a kind of transaction log of such changes, and have that replayed on the remote system rather than having Status Viewer send commands directly to the remote Bronx port.
- What kinds of delays are acceptable in synchronizing Primary and DR systems during normal operation?
- What kinds of atomic transactions will be used to synchronize Primary and DR systems?
- How will "which system is currently the Master Configuration Authority" be displayed on-screen to the Administrator? Answer: for a quick and dirty prototype, Roger has in mind the kind of top-of-screen message we use for license violations. If we do that, perhaps we need a different background color to ensure the users don't have message fatigue and overlook some part of this information. Longer term, in a product solution, we might display a clear omnipresent flag rather than a message, and also display full DR state information in a portlet on some Administration page.
- If communication breaks down between Primary and DR system, but each is still alive, then each will come to believe that it still has Master Configuration Authority. How do we systematically prevent this, or at least warn the administrator that the system might be in this condition so that manual (procedural) control may be applied in this situation? Answer: MCA is never transferred automatically. DR heartbeats, if they are able to get through, will detect such a configuration and generate alarms. Humans must be involved to assign and lift MCA capability at each end.

- Here are all the databases in a GW6.1 system:
  - GWCollageDB
  - dashboard
  - jbossdb (ignore this database — it contains only transient data)
  - jbossportal
  - logreports
  - monarch

  Which of these needs to be replicated to the alternate server, to what extent (full or partial, and which parts), on what schedule, and via what mechanism? Is shutdown of the entire Primary system necessary for such synchronization to occur safely? How long will such a shutdown take, and what are the side effects?
- Here are the extra databases added by NMS:
  - cacti
  - nedi
- What is the complete list of configuration files, outside of the databases, that need to be synchronized across the Primary and DR servers? (This needs to be listed on a per-application basis.)
- We need to understand the configurations we will be supporting with this product. Does it consist of only a Primary and a DR server, or will there be child servers involved? Answer: the prototype will have only Primary and DR server, with NMS on each. No GDMA and no child or proxy servers are involved. A future product release may add support for more complex configurations.
- Ask Kevin about heartbeat signals and he mechanism currently uses for them.
- What single points of failure still exist in our design?
- In what ways can we use passive safety mechanisms?
- How do we configure a Child server to send results of all types to two different places?
- How do we configure syslog-ng to send to two different places? How about syslog messages from other machines that might be configured to send to the monitoring servers? Examples are in order.

E.4 Prototype vs. Product

The initial prototype is limited in scope. Nonetheless, we wish to delineate the differences that might result from a step up in commitment to development.

E.4.1 Prototype Features

The features listed here are targeted to the initial customer delivery.

- simplified installation (e.g., tarball or perhaps RPM)
- command-line setup for most operations
- a few README-like text files for documentation
- support for only certain components (at this stage, basic GW Monitor plus NMS on a single server) and transformations
- no defined support for upgrades of GW Monitor or NMS; would need to re-apply this package afresh on both ends, after an upgrade

E.4.2 Product Features

Features listed here would be desired to round out the prototype into a supportable product.

- standardized packaging (e.g., a Bitrock installer)
- a DR-status portal enclosing a dynamic picture (who is up/down, which connections are present/broken, where is Notification Authority and Master Configuration Authority currently located, etc.)
- integration with the GroundWork Monitor license keys

- full integration with versioned (timestamped/annotated) configuration backups, including user-initiated selection and installation of an alternate configuration to run
- formal logging of the DR Engine actions and exceptions
- pretty and convenient UIs available for most operations
- support for extended GW Monitor configurations (e.g., Child servers; NMS on a separate server; multi-Cacti NMS setups)
- support for additional components (fping and other add-ons) and transformations (TBD)
- formal documentation
- support for upgrades of GW Monitor and add-on products
- full integration of the kinds of custom extensions we have previously done related to Standby, needed by customers such as Quantix and Viasat
- long-term product feature: support for a Staging Server, to make it easier to test configuration changes well before rolling new setups into production
- extend all third-party components to recognize when they are operating in Backup mode (i.e., without MCA), and prohibit their respective configuration changes under that condition
- extend all GroundWork and third-party components to tell when replication is in progress, and hold off allowing changes until replication is complete
- full config-file input validation, via a YAML::Valid module, something like XML DTD validation:
  - structure of document or node (mandatory/optional/prohibited keys)
  - value types (scalar/hash/array)
  - value limits (min/max; equal-to/must-be [for numeric/strings]; must-match [pattern or enumeration])
  - callbacks for more extensive (non-static) validation, such as calling an external program to perform parts of it
- choice of using SSL at the database level during replication (as opposed to either going without, or assuming the use of an encrypted tunnel to provide the desired security level)
- more attention paid to performance (e.g., setup for splitting database bin-log and relay-log files onto separate spindles, to avoid constant head chatter)
- extended monitoring of replication status
- handling of Primary and DR system timezone differences
- security improvements (authentication/authorization, data tainting, etc.)
- more data transforms available, perhaps
- GUI client for user interaction and status display (Tk, or possibly HTML)
- more robust error handling and logging
- more robust/automated error detection and handling (to deal with various conditions described in Section 16 of the MySQL Reference Manual on Replication)
- formal upgrade support during GW and DR product upgrades (preservation of local configuration, for instance)
- more extensive integration with backup/restore operations for all components; force on-demand backup/restore operations available, on either Master or Slave side
- ability to treat the DR Server as an offline Staging Server for testing new setups; then reverse roles and roll them out as a *fait accompli* to production on the Primary server
- handle partial NMS installations (not all components)
- formal coherence points (proper recognition of when sets of files are or are not in a consistent, useable state)
- extending third-party applications to recognize when we want them to treat configuration data as read-only, and having them obey such a restriction

- internal refactoring to improve implementation choices made due to early time pressure
- allow # comments in “recover”, to document scripts; also implement certain conditional testing and looping and sleeping, iteration counting, and loop break, to allow more sophisticated scripting such as replicating all objects and waiting the proper amount of time between successive steps
- provide full shell-like previous-command callback and editing in the “recover” program; is there some POE-related module that might provide such history editing?
- more extensive number of categories used in logging, for severity, monitor status, operation status; Roger has ideas on what might be useful in this area
- soft-landing shutdown available, so any replication actions currently in progress are allowed to complete (leaving the object up and operating) before the Replication Engine shuts down
- soft-block to do the same with respect to allowing current ongoing operations to fully complete, without a full shutdown of the Replication Engine
- support more general topologies than just a single pair of servers to replicate between
- provide a command to list out application and database aliases, all or individually (probably already provided with the “alias” command)
- more-complete backup management, including perhaps keeping only a limited number and/or time period of accumulated backups (probably already provided, as the system would be unmaintainable without such automation)
- better initialization/upgrades/validation of previous replication state
- synchronous, interruptible user-initiated commands with incremental output to the user terminal for long-running commands
- better notification of sync command execution errors back to the user terminal
- define a special Application Type for DR, to enable easier filtering of messages arising from this facility In the Event Console (shorter Application Type name? ask Roger)
- configurable color for “recover” prompt, so it stands out better
- “set autohelp on/off” --- print help section for commands given without arguments
- linked applications for coordinated actions during sync operations (various forms of dependency)
- Primary and DR systems must be time-synchronized?
- highlight errors and notices shown by “recover”, either in reverse video or in color, optionally if configured; use Term::Visual or some such package to help support this? or some Perl Termcap package?
- cron job replication, too
- full cron-style scheduling (or some other forms of syntax)
- continuous database replication (mostly for monarch, as we need snapshots for all the other databases anyway, as they are not suitable for continuous replication for one reason or another)
- handle garbage on the server sockets (close and re-open) to survive a security scan; test with telnet
- perhaps separate the frequency of replicating from the frequency of bouncing the slave to pick up changes, except when transitioning into failure mode; but this may be a bad idea, as you may only discover failure at the worst possible time
- implement the “sync” command to force scheduling of replication actions, in a safe manner (taking into account shared dependencies), so that changes made to a DR system during an extended outage can be quickly copied back to the Primary system and it can resume its usual role
- be more discriminating about what to send (compare working with replica: only send differences); have an application diff script to compare to see whether the application needs deploy or not, or whether the application must be bounced to capture changes
- better interlocking of the Replication State Engine with ctl.sh actions, so if the administrator intentionally brings down some component without first separately telling Replication about it, then Replication will recognize that fact and temporarily block replication actions from deploying or starting the object in question

- don't allow manual component startup to occur while Replication is in the middle of capture or deploy actions
- if the Replication State Engine is partway through a deploy operation and stopped there, that fact should be available to the object start routine so it can force either a rollback or commit operation to get back to a working configuration with the least amount of collateral damage
- GroundWork Monitor status probing (via ctl.sh) should reflect DR actions as well
- possible alternative to nagiosstatus.sav + bouncing nagios
- what HA-like capabilities might be useful in a DR context? things like network switch settings (NAT translations), with auto-switchover control? re-mount shared storage on a different machine?
- Implement the ability to release Notification Authority Control on a predetermined schedule, as described earlier in this paper. Also implement configuration settings to suppress the usual heartbeat-analysis warnings during such time periods, and (alternatively) either permanently (e.g., when the system has not yet been brought into full production because it is not yet fully set up) or temporarily (e.g., during a scheduled maintenance period). Such configuration settings and control within DR are necessary because it continually overrides whatever notification enable/disable setting is established within Monarch.

Appendix F: Development Stages, Tasks, and Estimation

The description here is currently only for development of prototype features.

F.1 Replication Control Engine

xxx

F.1.1 Heartbeat Scripting

- handle startup-time hold-down, to allow both ends to come up before alarming
- handling ongoing periodic probes
- handle link testing (distinguish a failed link from a failed peer)
- handle link-failure retries (to suppress short-duration transient failures unless they are too frequent; generate warnings, not failures, during this period)

F.1.2 Peer Health-Check Scripting

xxx

F.1.3 Scripting to replicate the `jbossportal` database

xxx

F.1.4 Scripting to replicate individual `GWCollageDB` tables

xxx

F.1.5 Simple file replication

xxx

F.2 Status Viewer Changes

The Status Viewer must be changed to:

- Accept a configuration of where the DR system Bronx port is located.
- Submit scheduled downtime, comments, and external commands not only to the Primary system Bronx port, but also to the DR system Bronx port.
- Report to the administrator any failure to send scheduled downtime, comments, and external commands to the secondary Bronx port.

F.3 Bronx Changes

The code behind the Bronx Command Acceptor port needs to be infused with the same restructuring bug fixes that were previously applied to the Listener port code.

F.4 Documentation

xxx

F.5 Packaging

xxx

F.6 Testing

xxx

F.6.1 Installation Mode

xxx

F.6.2 Normal Mode

xxx

F.6.3 Maintenance Mode

xxx

F.6.4 Failover During Normal Mode

xxx

F.6.5 Failover During Maintenance Mode

xxx

F.6.6 Failure Mode

xxx

F.6.7 Switchback

xxx

Appendix G: References

http://en.wikipedia.org/wiki/Disaster_recovery

http://en.wikipedia.org/wiki/High_availability

http://en.wikipedia.org/wiki/Service_Availability_Forum

http://en.wikipedia.org/wiki/Seven_tiers_of_disaster_recovery

http://www.opensaf.org/

http://www.saforum.org/

[DR Slides] http://licensing.steeleye.com/support/papers/lwsf_2004_slides.pdf
This presentation contains the best definitions I have see of DR and HA and the distinctions between them. Highly recommended reading.

http://tldp.org/HOWTO/NFS-HOWTO/security.html
A look at NFS security under Linux. Dated 2002, so the material may be old and need verification with current releases, but it at least identifies typical issues in detail. Many other resources on the Internet and in your own OS documentation should be consulted before setting up Secure NFS.

Fault Tolerance in GroundWork Monitoring Solutions: Best Practice Guide
A GroundWork Open Source white paper.

Appendix H: Revision History

The following versions of this document have been issued to date.

Table 3: Document Versions

Version	Date	Description of Changes
0.2.3	Mar 5, 2010	Initial rough-cut draft publication. Apologies for the lack of clean flow in some sections and most of the appendices. Also, changes to reflect the actual implementation are not yet folded in.
0.3.0	Jul 5, 2010	Current draft-in-progress. Probable last version of this document before it is split into multiple separate documents targeted toward separate audiences.