Call Resolver for HA: Design Specification

Revised: 4/16/06

Introduction

The initial Call Resolver only supports single-server configurations, where there is a single forking proxy and authorization proxy on the same machine as the Call Resolver. With sipX 3.4, the Call Resolver must support HA (highly available) configurations with distributed proxies. Note that HA applies to basic calling service, not to the Call Resolver itself, which remains a single point of failure for CDR generation.

HA terminology:

- The “master server” has a full set of services, including the two proxies, registrar, config server, and Call Resolver.
- The “distributed server” has just the proxies and the registrar required for basic calling.

This memo describes the design of Call Resolver changes required to support HA. See the Call Resolver design spec for background information.

Design

Logging Events

With distributed proxies, call state events (CSEs) now occur on multiple servers. The events for a single call may even be spread across multiple servers. How does the Call Resolver collect those events for analysis? Here are three options:

1. Proxies log events to a **local DB** on the server. Call Resolver queries each remote DB for events when it runs to get a combined event set for analysis.
2. Proxies log events to a **central DB** on the Call Resolver server. Call Resolver queries that central DB.
3. Proxies log events to XML **log files** locally. Call Resolver somehow retrieves that data from each distributed server when it runs.

Option #1 is a clear winner. Option #2 loses CSE data if the proxies lose their connection to the central server, or requires some mechanism to prevent data loss (such as logging locally!). Option #3 is more complicated to implement and doesn't leverage our investment in logging to a DB. The only downside of option #1 is that it requires a database (PostgreSQL currently) to be installed on the distributed server, but that is acceptable given the resulting benefits.

Database Instances

We considered splitting the SIPXCDR database (instance) into separate databases, one for CSEs and one for CDRs, since database on the proxies will only hold CSEs. See XPB-560. However, doing so would add too much complexity relative to the benefits, so we have dropped the idea.

Configuration

Call Resolver

Call Resolver previously loaded CSE data just from the local SIPXCDR database. Now it loads data from multiple databases. Call Resolver must be configured with knowledge of those

databases. Add this line to callresolver-config.in:

`SIP_CALLRESOLVER_CSE_HOSTS` : *comma-delimited list of machine names or addresses where CSE DBs are located*

See XPB-562 for more details.

Database

Configure PostgreSQL on the distributed server to accept remote connections from Call Resolver on the master server. See XPB-563.

Call Resolution

Event Order

Call Resolver currently analyzes events in sequence by timestamp. The multiple servers of an HA configuration are required to have synchronized clocks, but the clocks will never be perfectly synchronized. So Call Resolver needs to change to rely primarily on the SIP CSEQ value instead to determine event ordering. CSEQ is now included in call state events. Note that SIP 200 OK and ACK messages carry the same CSEQ as the original INVITE, so call request events (based on the INVITE) and call connected events (based on the 200 OK) will carry the same CSEQ.

Call Resolver logs a warning if it detects that the clocks are poorly synchronized. See XPR-145 and XPR-146.

Implementation Notes

- Currently each class connects to at most one database, e.g., `Gateway` connects to `SIPXCONFIG`. Figure out how to connect the same class, `CallStateEvent`, to different databases in sequence. Maybe we just make multiple `establish_connection` calls, but if so then what happens to objects in memory that were linked to the previous database?
- Since CSE queries are now networked rather than local, the overhead of querying the database has gone up, something to keep in mind when designing for performance (see XPR-144).