Study Case - Building Geo-Distributed Platforms with OpenSIPS

Vlad Paiu
OpenSIPS Project Developer
OpenSIPS Solutions

Outline

- From Single Server to Local HA
- From single POP to Multiple POPs
- What Next ?
- Conclusions

Start Point

Single SIP Server Instance

- Limited Performance
  - Number of Subscribers
  - Number of Calls per second

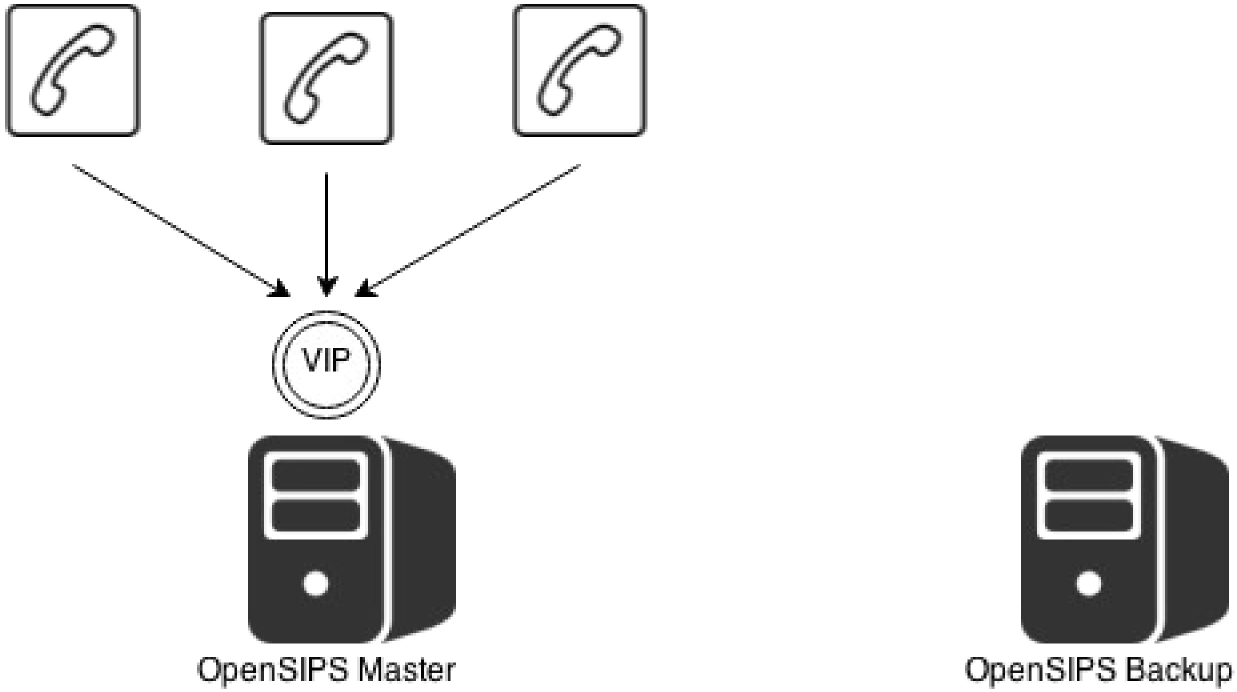
- Prone to Hardware Failures

From Single Server to Local HA

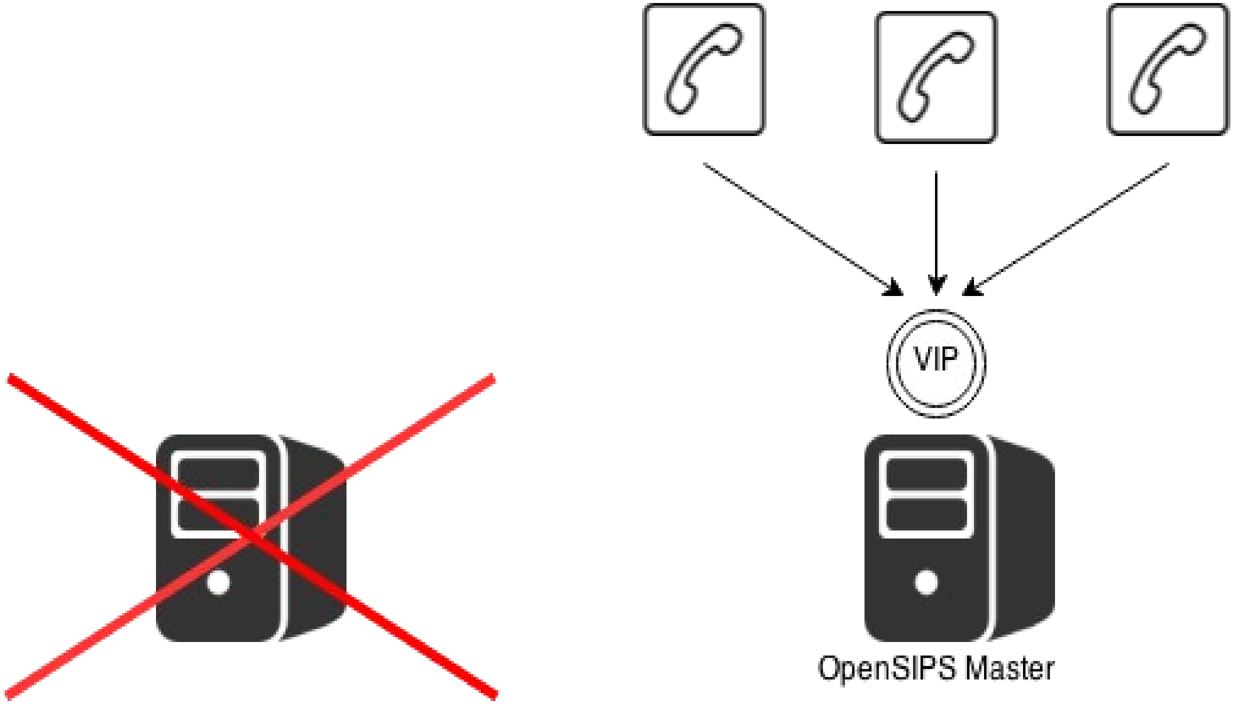
Hot Backup Local HA

- Use a Virtual IP
  - VRRP
  - HeartBeat
- The VIP has to be able to move
  - On Hardware Failures
  - On certain Software Failures / Patterns
  - On demand, when maintenance must be performed on a server
- HOT backup – OpenSIPS needs the VIP to start
  - Keep the VIP on both machines & deploy IPTables and ARPTables rules on backup
  - Keep VIP just on the Master machine and use /proc/sys/net/ipv4/ip_nonlocal_bind

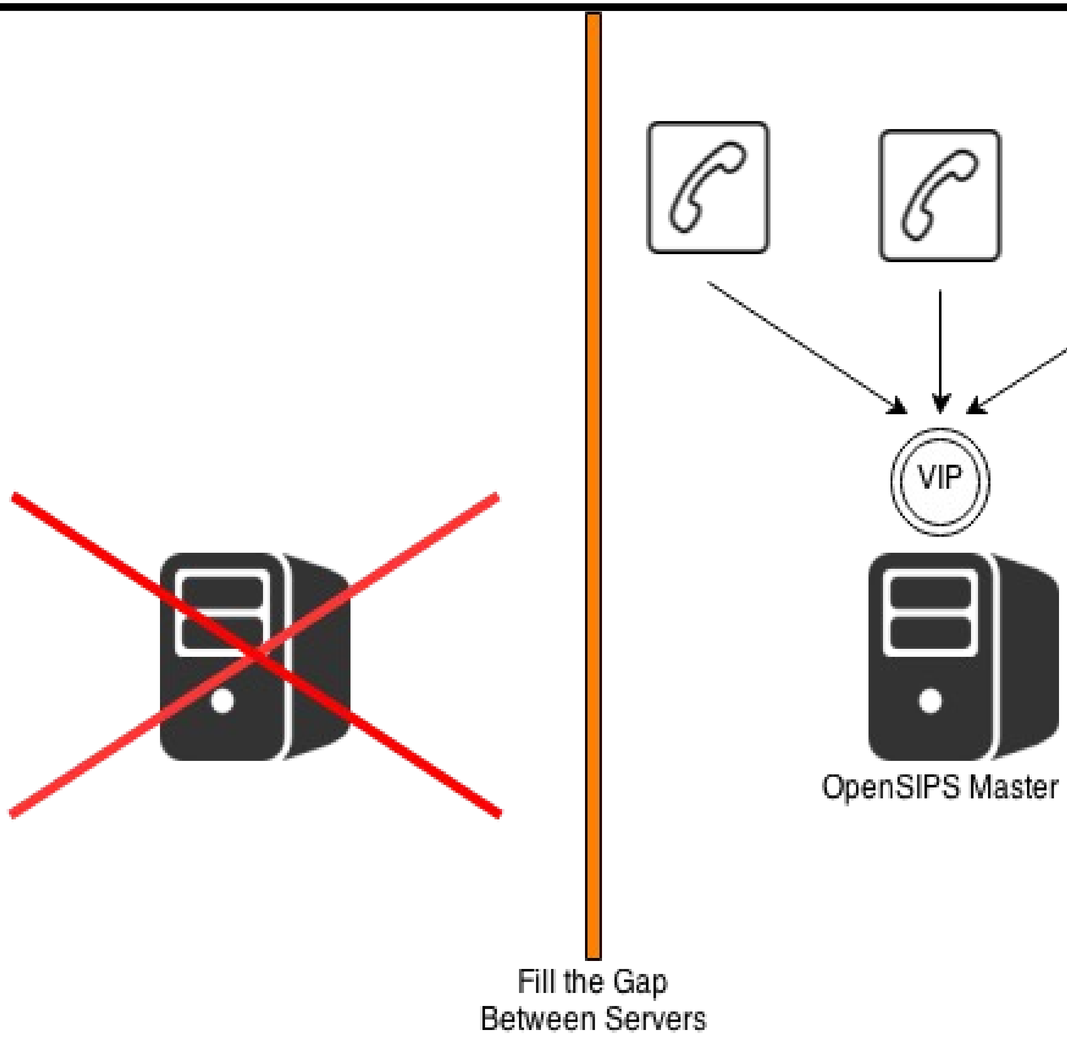
Hot Backup Local HA



Hot Backup Local HA



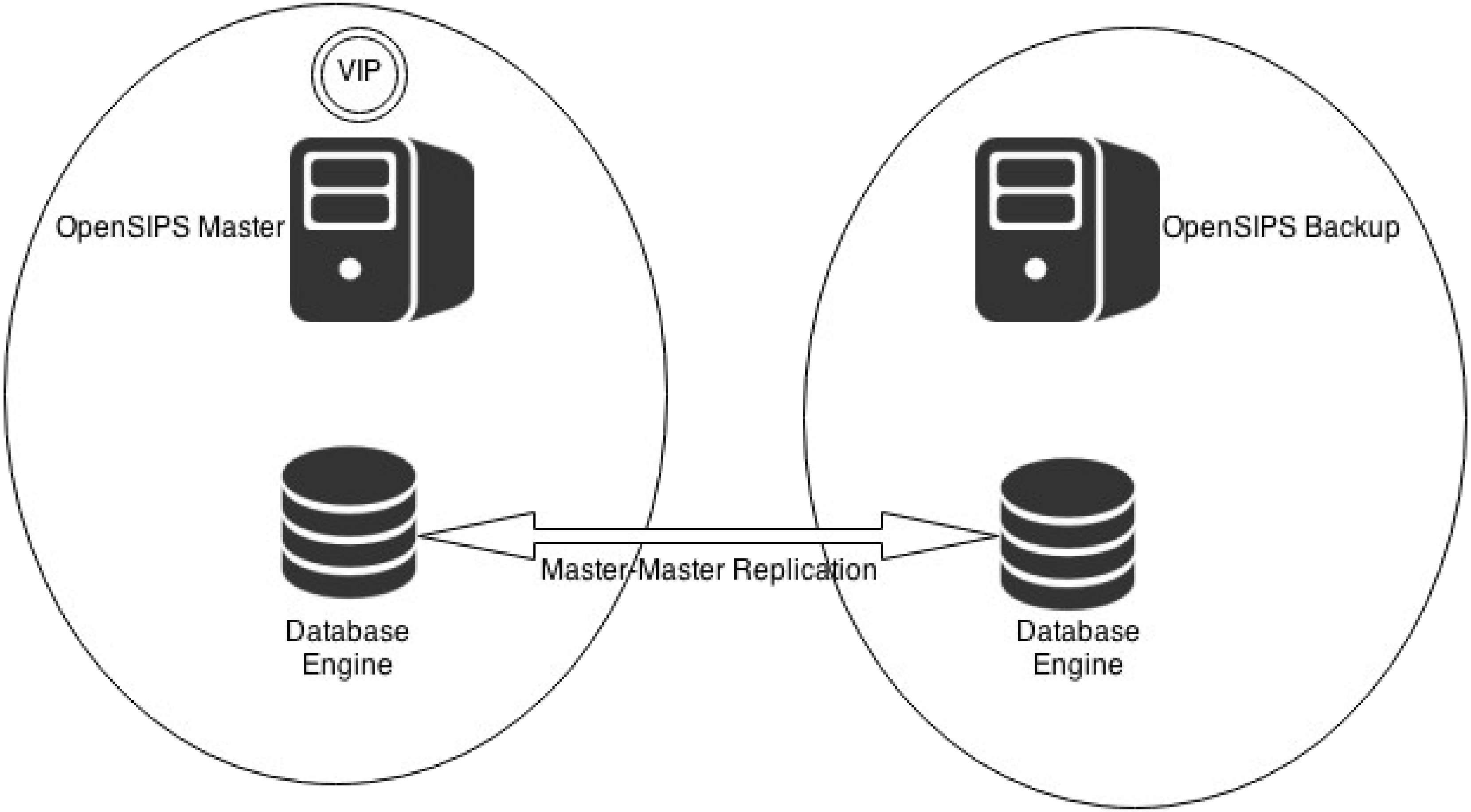
Hot Backup Local HA



Hot Backup Local HA

- Replicate Provisioning Data
- Replicate Runtime Data

Replicate Provisioning Data



Replicate Runtime Data

- Registrations
- Active Dialogs
- Before OpenSIPS 1.10, using the DB was the only way
  - Limited in terms of Performance
  - Slow fail-over

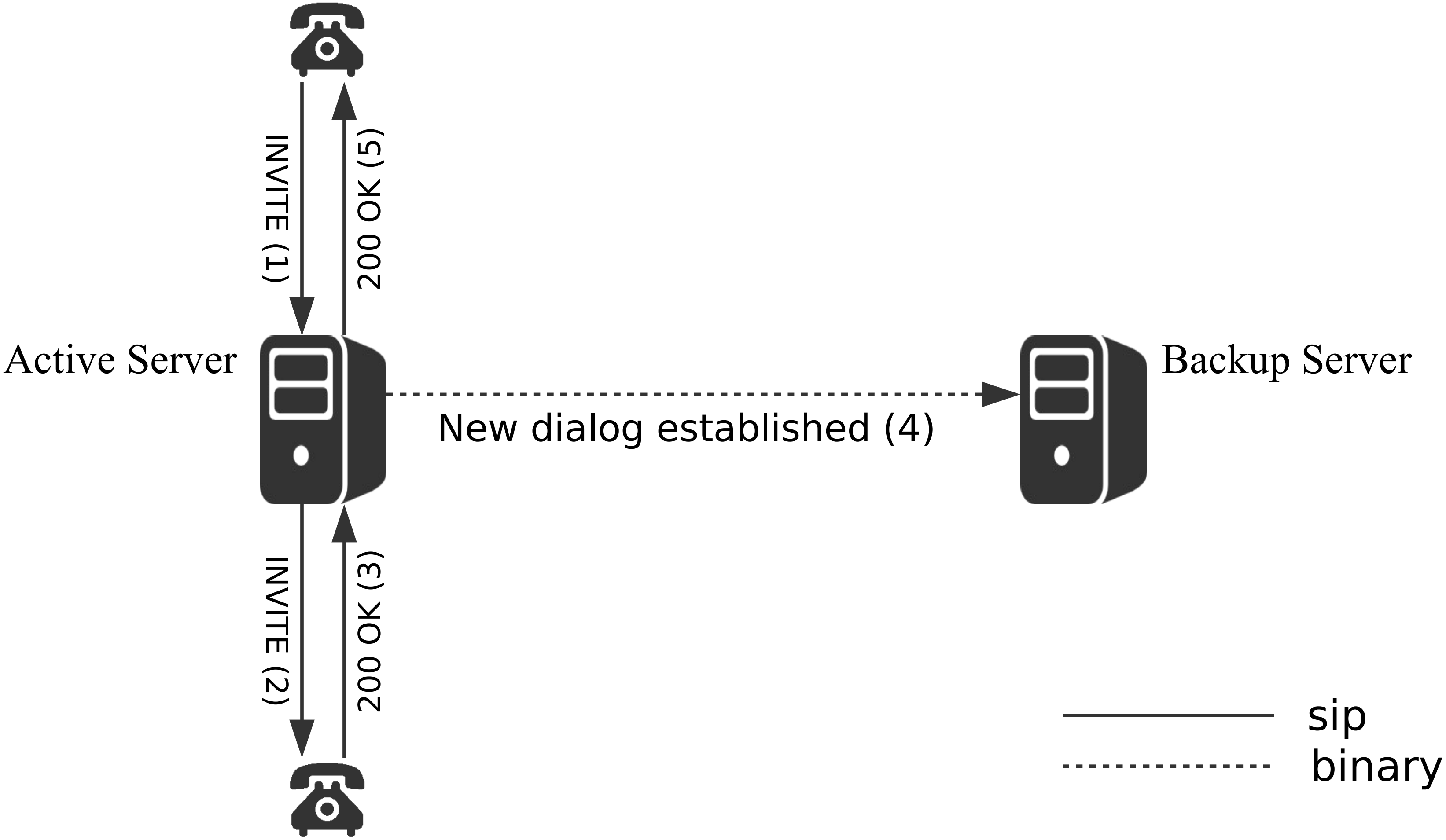
Binary Internal Interface

- Fast and Efficient communication channel between OpenSIPS instance

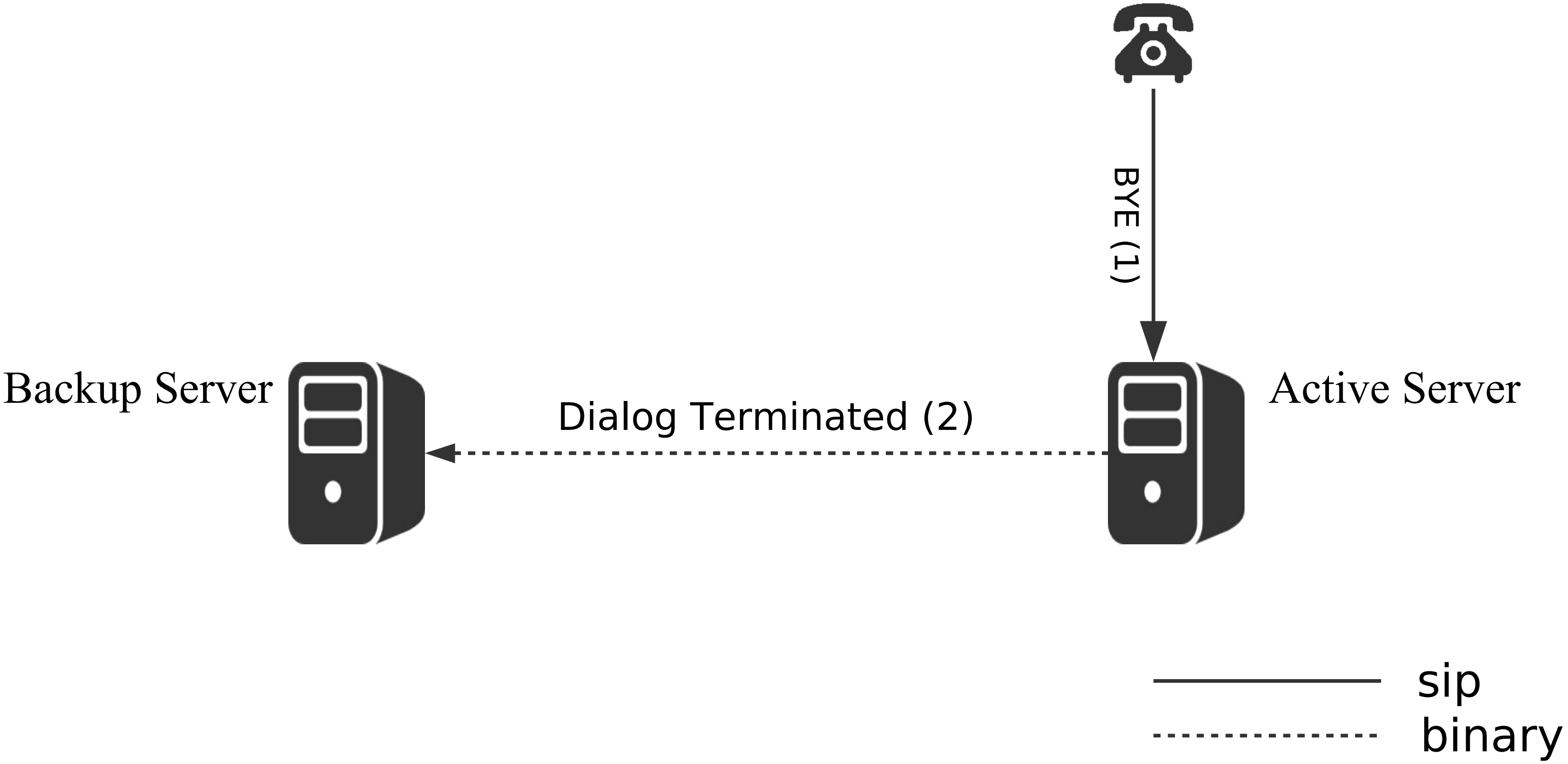
- To be used for real-time data replication
  - Dialog state
  - Registrations
  - Transactions

- Advantages
  - Faster when compared to DB / SIP Replication
  - Automatic – no script changes
  - Generic one to many replication channel

How It Works


INVITE (1)
200 OK (5)
Active Server
New dialog established (4)
Backup Server
INVITE (2)
200 OK (3)
sip
binary

How It Works


BYE (1)
Backup Server
Dialog Terminated (2)
Active Server
sip
binary

Configuration

- bin_listen = 10.0.0.150:5062
- bin_children = 5

- modparam("dialog", "accept_replicated_dialogs", 1)

- modparam("dialog", "replicate_dialogs_to", "10.0.0.150:5062")
- modparam("dialog", "replicate_dialogs_to", "192.168.2.129:5060")

From single POP to Multiple POPs

Why Go Distributed?

Geographical (coverage, QoS)

- Cover the map
- Best quality everywhere

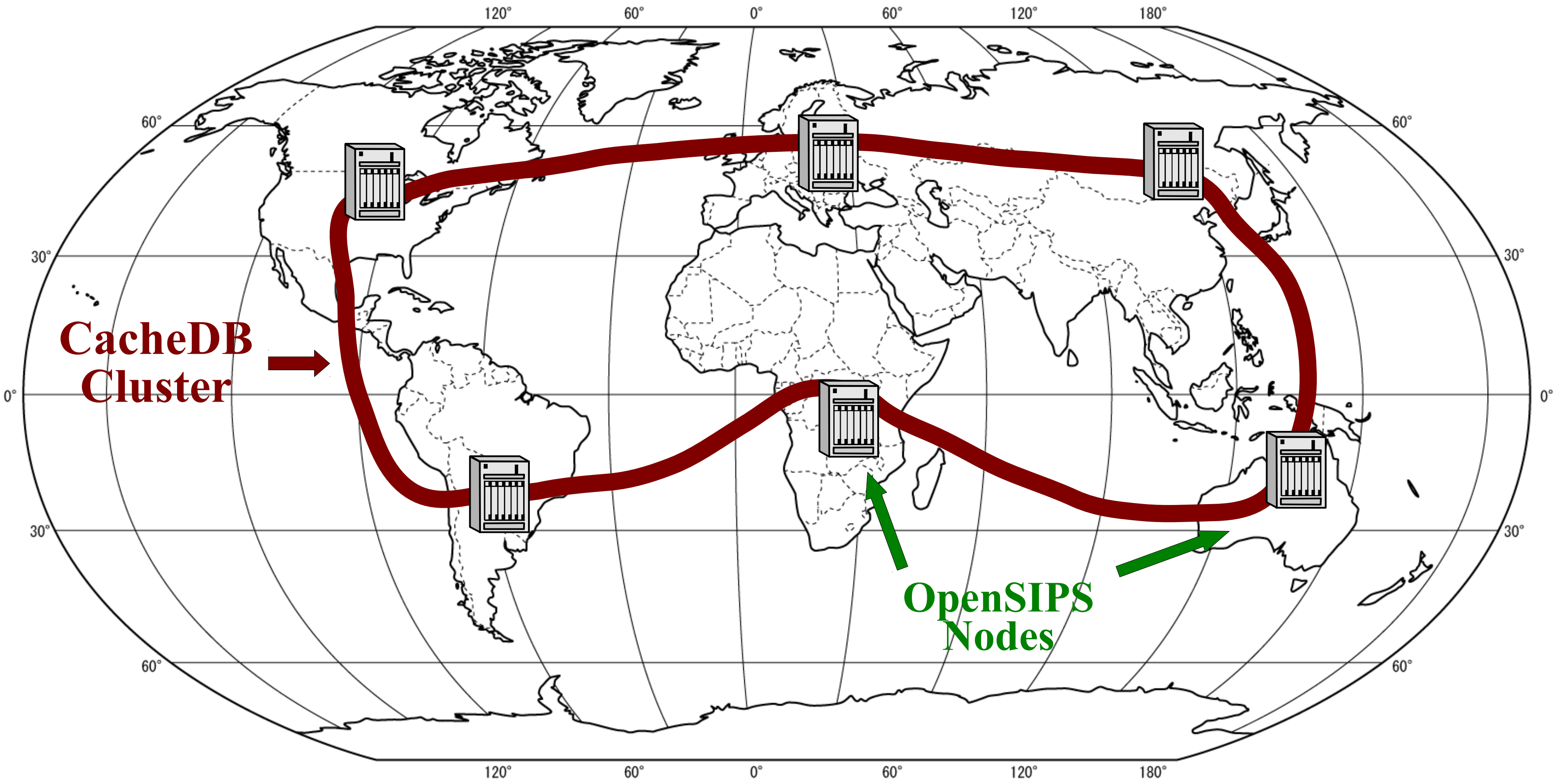
Load (balancing,scaling)

- Millions of concurrent calls
- Hundreds of millions of subscribers

Redundancy (HA)

- Having a backup is a good idea
- For the 5 of nines you need more than 100%

Distributed Architecture



Key Objectives

Multiple OpenSIPS instances located in geo-distributed POPs that act as a whole by sharing internal data (calls, registration, counters, statistics) or runtime external data (limits, credits, caches)

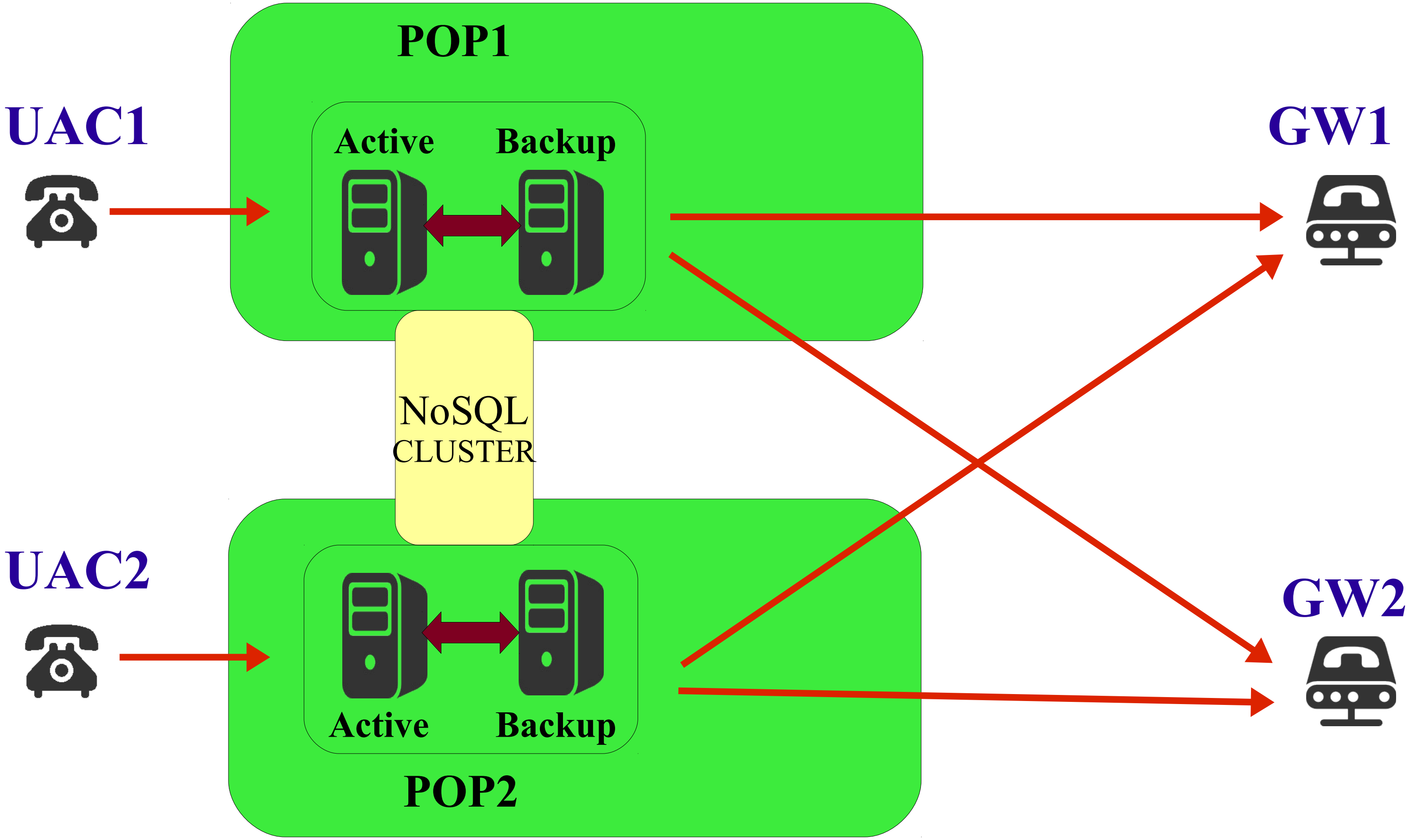
NoSQL Cluster

NoSQL DB support to link <u>everything together</u> - modules using the NoSQL interface to communicate and share information

NoSQL Support

- **Supported NoSQL Back-ends**
  - **Redis**
  - **Memcached**
  - **Cassandra**
  - **MongoDB**
  - **CouchBase**
- **Uniform access to NoSQL backends**
  - **Simple API like fetch_key(), remove_key(), etc**
- **Raw back-end specific queries**

Architecture



Access mode

- Drop Accessing directly via IP
- Rely on DNS
  - GeoIP DNS Location
  - SRV for Inter-POP Fail-over

Replicating Registration State

- Users can register / make calls from any POP
- A user can be contacted from any POP

- Instead of replicating the entire Registration State, we will replace just the actual POP where it is registered
  - Faster (less information to replicate)
  - For NAT, just the Registration POP can contact the user

Replicating Registration State

In the POP where the user Registers

- Use event Routes that are triggered when a new AOR is registered / unregistered and use the NoSQL cluster to store the AOR and POP

Replicating Registration State

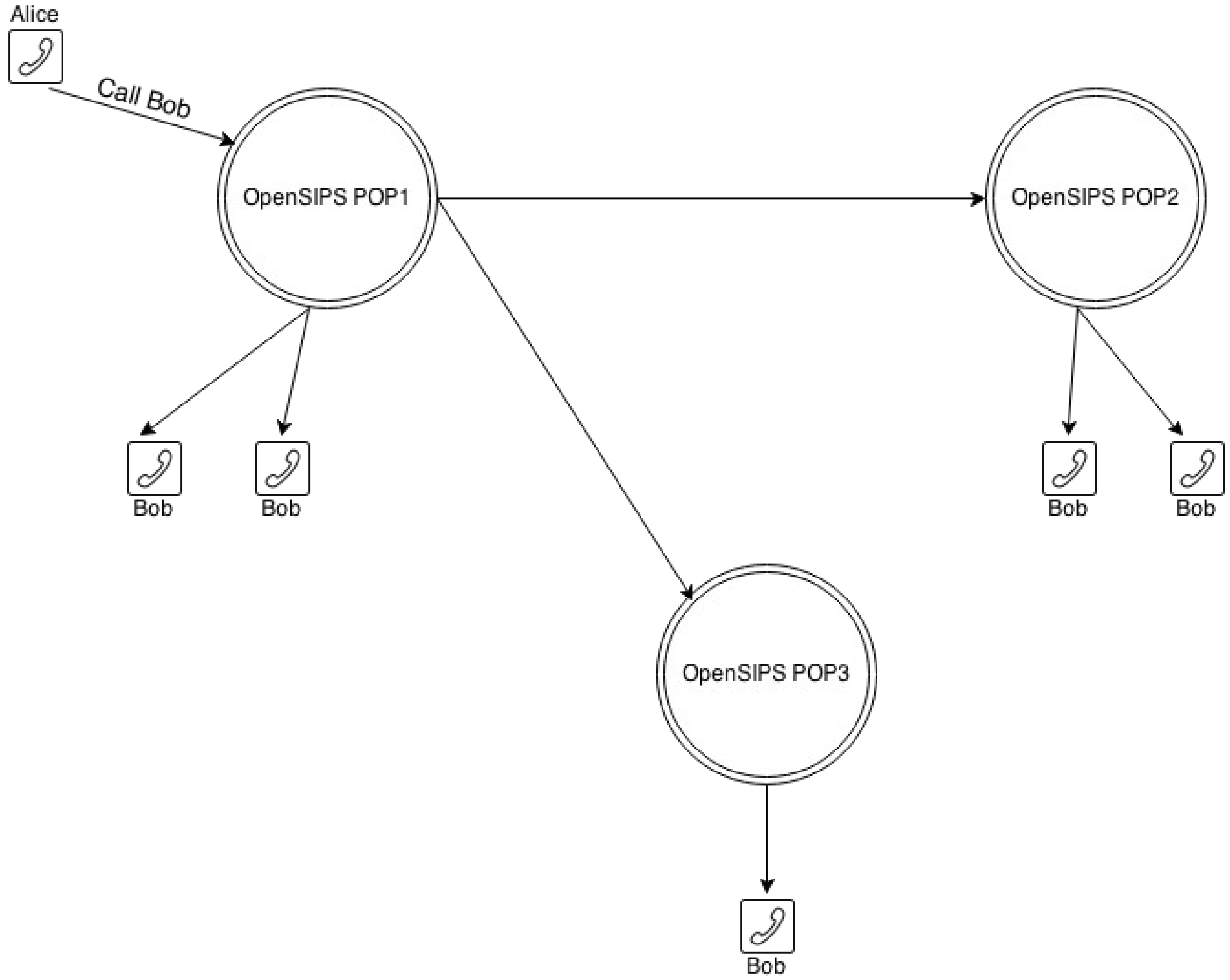
```
loadmodule “cachedb_mongodb.so”
loadmodule “usrloc.so”
loadmodule “registrar.so”
modparam(“cachedb_mongodb.so”,”cachedb_url”,”mongodb:location://cluster_host:cluster_port/UserLocation.UserLocation”)


event_route[E_UL_AOR_INSERT] {
    fetch_event_params(“$avp(aor)”);
    xlog(“SCRIPT:REG:DBG: new AOR $avp(aor) registered \n”);
cache_raw_query(“mongodb:location”,"{ "op” :"insert”,"ns” :"UserLocation.UserLocation”,
"query”: {"aor” : "$avp(aor)”, "pop”: "MY_POP_ID”} }”);
}
event_route[E_UL_AOR_DELETE] {
    fetch_event_params(“$avp(aor)”);
    xlog(“SCRIPT:REG:DBG: AOR $avp(aor) de-registered \n”);
cache_raw_query(“mongodb:location”,"{ "op” :"remove”,"ns” : "UserLocation.UserLocation”,
"query”: {"aor” : "$avp(aor)”, "pop”: "MY_POP_ID”} }”);
}
```

Replicating Registration State

In the POP where the user needs to be contacted

- **First lookup the user locally, and then also lookup the user in the NoSQL cluster, appending branches for each separate POP the user is registered in**
- **All the call handling is done on the originating POP, the destination POPs will just have to relay the call to the user.**

Inter-POP Routing



Replicating Registration State

```
$var(lookup) = lookup(“location”);
$var(aor) = $rU+"@"+$rd;
$var(rc) = cache_raw_query(“mongodb:location”,"{ "op” : "find”, "ns” :
"UserLocation.UserLocation”, "query”: {"aor” : "$var(aor)”},"$avp(mongo_result)”);
 while ($(avp(mongo_result)[$var(it)]) != NULL) {
     $json(json_res) := $(avp(mongo_result)[$var(it)]);
     $var(pop) = $json(json_res/pop);
     if ($var(pop) != “MY_POP_ID”) {
            $var(prev_du) = $du;
            $du = $var(pop_sip_url);
            append_branch();
            $du = $var(prev_du);
  }
```

Use the NoSQL cluster

- **Sharing concurrent calls counters**
  - modparam("dialog", "profiles_with_value", "caller/s”)
  - modparam("dialog", "cachedb_url", "redis://127.0.0.1:6379")
- **Generic Key-Value**
  - cache_store / cache_fetch
- **Generic Counters**
  - cache_add / cache_sub / cache_counter_fetch
- **Raw Back-end Queries**
  - Redis
  - MongoDB

Replicating Provisioning Data

- Try to have a Geo-Distributed SQL cluster
  - Painful
- Use the NoSQL Cluster
  - DB_CACHEDB module converts from SQL to NoSQL Queries
  - No script changes, no implementation change for the existing modules
  - Only MongoDB supported – for now

```
loadmodule "auth_db.so"


loadmodule "db_cachedb.so"
loadmodule "cachedb_mongodb.so"


modparam("db_cachedb","cachedb_url","mongodb://127.0.0.1/my_db.col")
modparam("auth_db","db_url","cachedb://mongodb")
```

What Next ?

Next Steps

- Optimize Media Paths
  - RTP Relay
  - Conferencing
  - Voice Mail
- CDR Propagation and Collection
- Save Bandwidth
  - SIP
    - Repackaging
    - Compression of Body / Not needed headers
  - RTP

Conclusions

- OpenSIPS is a great choice if you want
  - High performance on a single box
  - Strong geo-distribution capabilities

Thank you for your attention
You can find out more at www.opensips.org
vladpaiu@opensips.org

www.opensips-solutions.com

Questions are welcome