Corosync Cluster Engine: Designing High Availability

Steven Dake
June 2010

Agenda

- What is High Availability
- Project History
- Features
- Example Design
- Real-World Designs
- Quality and Closing

What is High Availability?

- Simple Equation:

$$A = \frac{MTBF}{MTBF + MTTR}$$

- MTBF = mean time between failures
- MTTR = mean time to repair
- A = probability system will provide service at a random time (ranging from 0 to 1)

What is High Availability?

$$A = \frac{MTBF}{MTBF + MTTR}$$

- Two ways to improve availability:
  - Increase MTBF to very large values
  - Reduce MTTR to very low values

High Availability is achieved through the <u>manipulation of MTBF and MTTR parameters</u> of system design to <u>meet</u> availability requirements.

Hardware Failure Cases

- Hardware Failure Causes:
  - Design failure (rare)
  - Random failure (rare)
  - Infant Mortality (high rate of failure)
  - Wear Out (high rate of failure)
- Increasing hardware MTBF:
  - Use better components
  - Preemptively replace hardware prior to wear out

Software Failure Cases

- Implementation Defects (very common):
  - Typically measured in defects per KLOC
- Increasing software MTBF:
  - Experienced engineering team
  - Peer review of all code
  - Simple design
  - Compact code foot print
  - Static and runtime analysis tools such as valgrind, lint, high compiler warning levels, coverity, lcov
  - Test coverage of the software

What about reducing MTTR?

- Older Models:
  - 24 hour monitoring and support staff
  - On-site spares
- Newer Models:
  - Depend on active redundancy
  - Stateless fail-over
  - N-way state replication with fail-over

Corosync Project History

- Started life as “openais.org” in 2002
- Announced Corosync in July 2008
- First 1.0.0 release in July 2009
- “flatiron branch” feature frozen in June 2010
- “weaver's needle” branch announced in June 2010

Features Overview

- Four C Programming APIs to create HA aware applications
- Ethernet and Infiniband ipv4/ipv6 Native Network Support
- Diagnostics and failure analysis
- 32/64 bit BE/LE support
- High focus on correctness and performance
- Network Security Services for authentication and encryption

Project Philosophy: Allow developers to create HA apps however they desire.

The Closed Process Group API

- Applications join a named group
- A group member may publish a message to all group members
- Messages are delivered asynchronously in atomic order to all nodes

The Simple Availability Manager API

- Initialize API called which generates a replica of the process
- If process fails because of defect:
  - SAM library generates a replica of the replica process
  - SAM starts the replica process
- Failures are detected between the parent replica process and the child active process via health-checking

The Configuration Database API

- Provides statistics and configuration information
- Permits applications to store data in the in-memory database (not replicated)

The Quorum API

- Our HA model allows for partitions
- Quorum provides notification to the application that the process may not continue because of a partition

Features: logging diagnostics

- High Performance Low Impact Failure Analysis
- Four logging targets
  - memory, stderr, syslog, file
- All events go to memory
  - HP Z800 Xeon 5530 single node consumes ~5% of the corosync process utilization running full cpgbench with 171,751,968 recorded events to memory as measured with oprofile and corosync-fplay
- Administrator configures which events go to other log targets

Features: logging diagnostics

- Example:

[root@cast sdake]# corosync
[root@cast sdake]# killall -SEGV corosync
[root@cast sdake]# **corosync-fplay** | tail -10
rec=[134] Log Message=Synchronization barrier completed
rec=[135] Log Message=Committing synchronization for (corosync cluster closed process group service v1.01)
rec=[136] Log Message=mcasted message added to pending queue
rec=[137] Log Message=releasing messages up to and including a
rec=[138] Log Message=Delivering b to c
rec=[139] Log Message=Delivering MCAST message with seq c to pending delivery queue
rec=[140] Log Message=Completed service synchronization, ready to provide service.
rec=[141] Log Message=releasing messages up to and including b
rec=[142] Log Message=releasing messages up to and including c
Finishing replay: records found [142]

Features: statistics diagnostics

- Configuration and Statistics Database populated with diagnostics information

```
[root@cast sdake]# corosync-objctl runtime.connections
runtime.active=1
runtime.closed=3
runtime.corosync-objctl:7595:10.service_id=11
runtime.corosync-objctl:7595:10.client_pid=7595
runtime.corosync-objctl:7595:10.responses=8
runtime.corosync-objctl:7595:10.dispatched=0
runtime.corosync-objctl:7595:10.requests=11
runtime.corosync-objctl:7595:10.sem_retry_count=0
runtime.corosync-objctl:7595:10.send_retry_count=0
runtime.corosync-objctl:7595:10.recv_retry_count=0
runtime.corosync-objctl:7595:10.flow_control=0
runtime.corosync-objctl:7595:10.flow_control_count=0
runtime.corosync-objctl:7595:10.queue_size=0
```

Example Design

- Sample program simulates the state of a savings account
- Any node may deposit or withdraw money from accounts in the system
- 32 nodes keep copy of all transactions in memory
- If application fails to health-check, it is restarted via SAM
- Savings account transactions replicated with CPG

main() - using SAM

```
static int instance_id;
static int healthy  = 1;

Int hc_callback (void *)
{
        If (healthy) {
                        return 0;
        }
        return -1;
}
int main (void)
{
        cs_error_t res;

        res = sam_initialize(2000, SAM_RECOVERY_POLICY_RESTART);
        res= sam_register(&instance_id);
        res= sam_hc_callback_register(hc_callback);
}
```

This simple code will execute the restart of the process if it fails to health-check

main() - Initializating cpg

```
static cpg_handle_t handle;

static void cpg_deliver_fn {
        cpg_handle_t handle,
        const struct cpg_name group_name,
        uint32_t nodeid,
        uint32_t pid,
        void *msg,
        size_t msg_len)
{
        /* Process messages here -
         Shown on later slide */
}

static struct group_name savings_group  {
        .value = “savings”,
        .len = 7
};
static cpg_callbacks_t callbacks = {
        .cpg_deliver_fn   = cpg_deliver_fn,
        .cpg_confchg_fn = NULL
};
```

```
int main (void)
{
        cs_error_t res;

        res = cpg_initialize(&handle, &callbacks);
        res = cpg_join (handle, group);
}
```

How to make messages

```
#define MESSAGE_ID_DEPOSIT      1
#define MESSAGE_ID_WITHDRAW     2

struct savings_header {
        uint32_t msg_id;
        uint32_t size;
};

struct savings_depost_msg {
        struct savings_header;
        char account[128];
        uint64_t pennies;
};

struct savings_withdraw_msg {
        struct savings_header;
        char account[128];
        uint64_t pennies;
        void *withdraw_failed;
};
```

How to send a Deposit

```
static cs_error_t deposit_send (char account_number[128], uint64_t pennies)
{
	struct savings_depost_msg depost_msg;
	struct iovec iov;
	cs_error_t res;

	deposit_msg.savings_header.msg_id = MESSAGE_ID_DEPOSIT;
	deposit_msg.savings.header.size = sizeof (deposit_msg);
	memcpy (&depost_msg.account, account, 128);
	depost_msg.pennies = pennies;
	iov.iov_base  = depost_msg;
	iov.iov_len = 1;

	res = cpg_mcast_joined (handle, CPG_TYPE_AGREED, &iov, 1);

	return (res);
}
```

How to send a Withdraw

```
static cs_error_t withdraw_send (char account_number[128], uint64_t pennies,
        void (*withdraw_failed) (void))
{
        struct savings_withdraw_msg depost_msg;
        struct iovec iov;
        cs_error_t res;

        withdraw_msg.savings_header.msg_id = MESSAGE_ID_DEPOSIT;
        withdraw_msg.savings.header.size = sizeof (deposit_msg);
        memcpy (&withdraw_msg.account, account, 128);
        withdraw_msg.pennies = pennies;
        withdraw_msg.withdraw_failed = withdraw_failed;
        iov.iov_base  = withdraw_msg;
        iov.iov_len = 1;

        res = cpg_mcast_joined (handle, CPG_TYPE_AGREED, &iov, 1);

        return (res);
}
```

main(): How to process pending messages

```
Int main (void)
{
        fd_set read_fds;
        Int select_fds;

        FD_ZERO (&read_fds);
        res = cpg_fd_get (handle, &select_fd);
        for (;;) {
                FD_SET(select_fd, &read_fds);
                If (FD_ISET (select_fd, read_fds)) {
                        res = cpg_dispatch (handle, CS_DISPATCH_ALL);
                }
        }
}
```

cpg_deliver_fn() - processing messages

```
static void deposit_process (const struct savings_header *hdr)
{
	const struct deposit_message = (const struct depost_message *)hdr;

	/*
	 * if depost_message->account found, add pennies to in memory storage
	 */
}

static void withdraw_process (const struct savings_header *hdr, int nodeid)
{
	const struct withdraw_message = (conststruct withdraw_message *)hdr;

	/*
	 * if deposit->message->account found
	 *	if account pennies > withdraw_message->pennies {
			Do withdraw
		} else {
			cpg_local_get (handle, &my_node_id);
			If (my_node_id == nodeid) {
	 *				withdraw_message->withdraw_failed();
			}
	 */
}
```

cpg_deliver_fn() - processing messages

```
cpg_deliver_fn (...) {
        const struct savings_header header = (const struct savings_header
*)msg;

        switch (msg->id) {
                case MESAGE_ID_DEPOSIT:
                        deposit_process (msg);
                        break;
                case MESSAGE_ID_WITHDRAW:
                        withdraw_process (msg, nodeid);
                        break;
        }
}
```

Merging State after a Partition

- Determine inconsistent state by message exchange
- Make state consistent on all nodes via message exchange
- Watch for my Linux Symposium Paper for more details on merging state

Real Design: Pacemaker

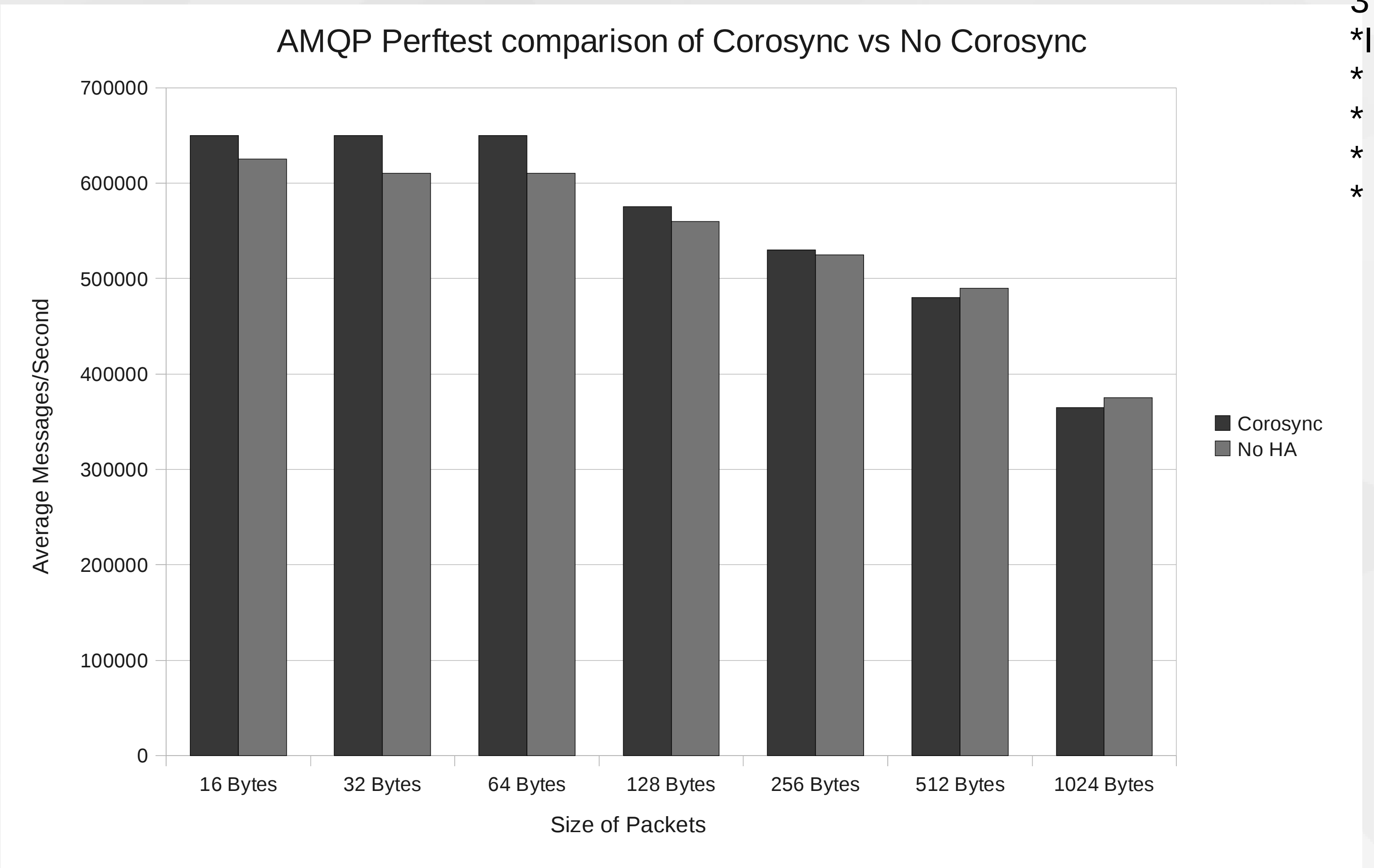
- Pacemaker is an Availability Manager for non-ha-aware applications
- Community chosen standard AM going forward for most Linux Distributions

Supports stateless failover model

Real Design: Apache QPID

- QPID Implements the AMQP messaging standard with very high performance requirements
- QPID is a ha-aware application
- All AMQP messages are replicated to multiple nodes in the Corosync Cluster

Example Design
QPID Benchmarks over IBA



3 node cluster
*IBM X3550
* 2 cpu 4-Core Xeon
* E5420 2.5 ghz
* 16 GB Ram 266mhz
* Mellanox MT25204

Corosync Quality

- Average professional engineering experience for major contributors is 12 years
- All patches require peer-review from a community member
- Test suite with 91 test cases run against each commit to the tree automatically
- Use of valgrind, coverity, lint, lcov regularly
- Identify testing gaps via lcov and develop new test cases to verify code that doesn't have coverage
- Compact code base – flatiron sloccount shows 42k lines of code
- Available in nearly every modern Linux distribution – lots of eyeballs, platforms, environments to find defects

Closing

- Corosync is likely already in your Linux distribution but if not, Download today:

  http://www.corosync.org

- Test Coverage:

  http://www.corosync.org/testcoverage

- Automated Builds and Testing:

  http://www.corosync.org:8010