# Sustaining CTDB Development

Amitay Isaacs
amitay@samba.org

Samba Team
IBM (Australia Development Labs, Linux Technology Center)

# CTDB Project

**Motivation:** Support for clustered Samba

- Multiple nodes active simultaneously
- Communication between nodes (heartbeat, failover)
- Share databases between nodes

**Features:**

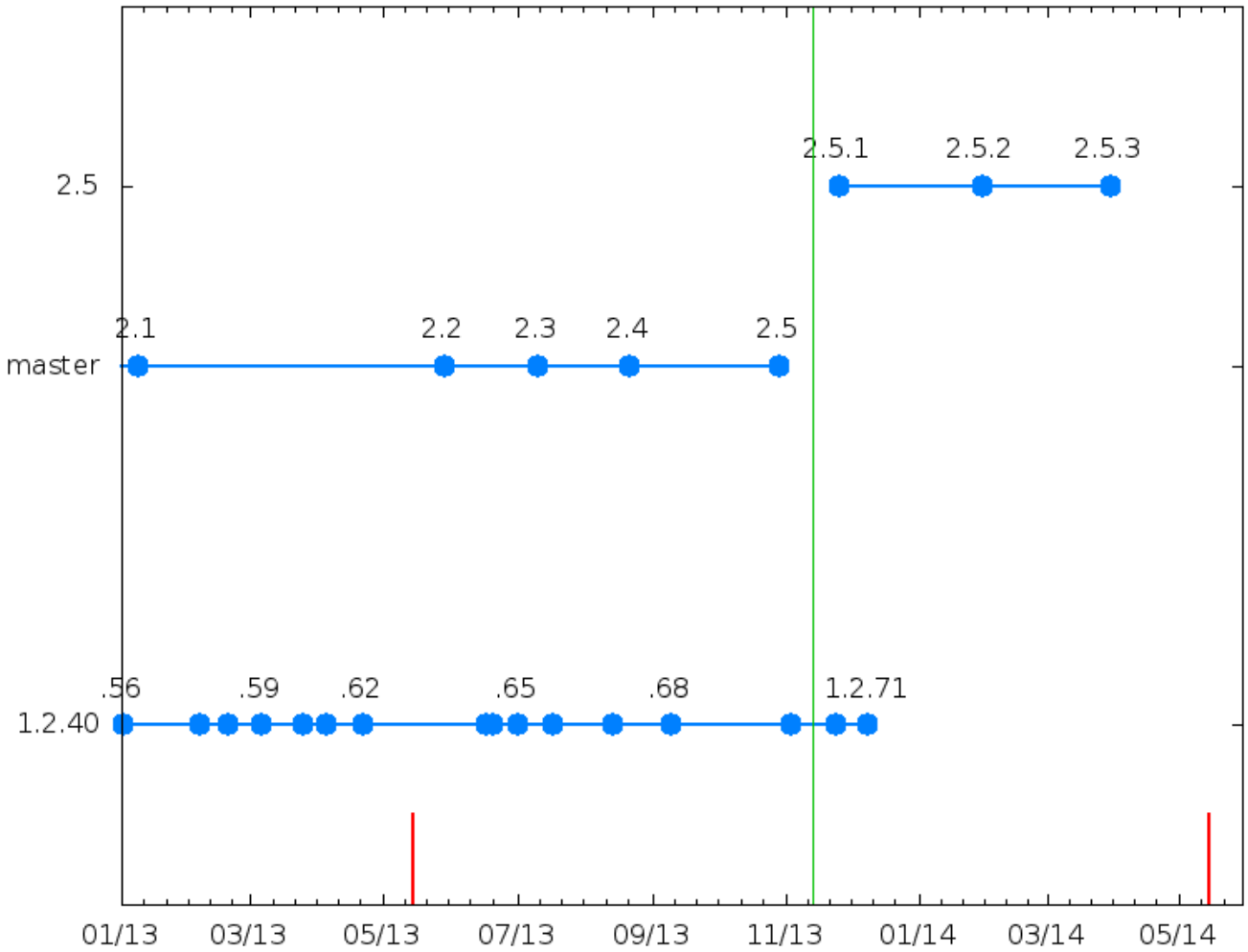
- Volatile and Persistent databases
- IP failover and load balancing
- Service monitoring

**Community:**

- `http://ctdb.samba.org`
- `git://git.samba.org/ctdb.git`,
  `git://git.samba.org/samba.git`

# Current Status

# Branches & Releases



# CTDB Releases

- 2.2 (May 2013) – 233 patches
  - performance improvements
  - recovery/vacuum database corruption fixes
  - fix race conditions in ctdb tool
- 2.3 (July 2013) – 120 patches
  - Add systemd support, fixes to banning code, improved traverse
- 2.4 (August 2013) – 90 patches
  - Improved ctdb startup sequence, socket handling
  - Fixed flags handling in recovery daemon, vacuuming bugs
- 2.5 (November 2013) – 146 patches
  - Moved ctdb socket from /tmp/ctdb.socket, change default dir
  - Improved documentation

# CTDB Releases

- 2.2 (May 2013) – 233 patches
  - performance improvements
  - recovery/vacuum database corruption fixes
  - fix race conditions in ctdb tool
- 2.3 (July 2013) – 120 patches
  - Add systemd support, fixes to banning code, improved traverse
- 2.4 (August 2013) – 90 patches
  - Improved ctdb startup sequence, socket handling
  - Fixed flags handling in recovery daemon, vacuuming bugs
- 2.5 (November 2013) – 146 patches
  - Moved ctdb socket from /tmp/ctdb.socket, change default dir
  - Improved documentation
- CTDB tree merged with Samba

# CTDB Releases

- 2.5.1 (November 2013) - 47 patches
  - Per database locking limits, vfork for locking children
  - Fixes to ctdb tool, persistent transaction code
- 2.5.2 (January 2014) - 36 patches
  - Fix ctdb reloadips
  - Event scripts run with vfork
- 2.5.3 (March 2014) - 130 patches
  - Improvements to vacuuming performance
  - Record locking compares hashes instead of keys

# Developers

**Contributions in 2013 - CTDB tree**

| | |
|---|---|
| 380 | Martin Schwenke |
| 233 | Amitay Isaacs |
| 46 | Michael Adam |
| 13 | Mathieu Parent |
| 6 | Sumit Bose |
| 4 | Volker Lendecke |
| 2 | Srikrishan Malik |
| 1 | Christian Ambach |
| 1 | David Disseldorp |

# Developers

**Contributions since Jan 2014 - CTDB tree**

| | |
|---|---|
| 75 | Martin Schwenke |
| 57 | Amitay Isaacs |
| 43 | Michael Adam |
| 3 | Srikrishan Malik |
| 1 | David Disseldorp |

# Developers

**Contributions since Jan 2014 - Samba tree**

| | |
|---|---|
| 62 | Martin Schwenke |
| 44 | Amitay Isaacs |
| 37 | Michael Adam |
| 3 | Srikrishan Malik |
| 2 | Gregor Beck |
| 1 | Andrew Bartlett |
| 1 | Björn Baumbach |
| 1 | David Disseldorp |
| 1 | Matthias Dieter Wallnöfer |
| 1 | Volker Lendecke |

# CTDB merge with Samba

# CTDB merge with Samba

- Motivation
  - Remove duplication of talloc, tdb, tevent, replace libraries
  - Autobuild testing of clustered Samba
  - Leverage off Samba release process

# CTDB merge with Samba

- Motivation
  - Remove duplication of talloc, tdb, tevent, replace libraries
  - Autobuild testing of clustered Samba
  - Leverage off Samba release process
  - Attract more developers

# CTDB merge with Samba

- Motivation
  - Remove duplication of talloc, tdb, tevent, replace libraries
  - Autobuild testing of clustered Samba
  - Leverage off Samba release process
  - Attract more developers
- Not there yet!

# CTDB merge with Samba

- Motivation
  - Remove duplication of talloc, tdb, tevent, replace libraries
  - Autobuild testing of clustered Samba
  - Leverage off Samba release process
  - Attract more developers
- Not there yet!
- To Do

# CTDB merge with Samba

- Motivation
  - Remove duplication of talloc, tdb, tevent, replace libraries
  - Autobuild testing of clustered Samba
  - Leverage off Samba release process
  - Attract more developers
- Not there yet!
- To Do
  - Create waf build for CTDB, Clustered Samba

# CTDB merge with Samba

- Motivation
  - Remove duplication of talloc, tdb, tevent, replace libraries
  - Autobuild testing of clustered Samba
  - Leverage off Samba release process
  - Attract more developers
- Not there yet!
- To Do
  - Create waf build for CTDB, Clustered Samba
  - Setting up clustered samba instance for autobuild

# CTDB merge with Samba

- Motivation
  - Remove duplication of talloc, tdb, tevent, replace libraries
  - Autobuild testing of clustered Samba
  - Leverage off Samba release process
  - Attract more developers
- Not there yet!
- To Do
  - Create waf build for CTDB, Clustered Samba
  - Setting up clustered samba instance for autobuild
  - Split monolithic code

# Bugs

# Tevent bug

## Problem

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

# Tevent bug

**Problem**

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

- Instrumented tevent epoll loop to debug timing issues

# Tevent bug

## Problem

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

- Instrumented tevent epoll loop to debug timing issues

```
+    if (getpid() == ctdbd_pid) tevent_before_wait(epoll_ev->ev);
     ret = epoll_wait(epoll_ev->epoll_fd, events, MAXEVENTS, timeout);
+    if (getpid() == ctdbd_pid) tevent_after_wait(epoll_ev->ev);
```

# Tevent bug

## Problem

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

- Instrumented tevent epoll loop to debug timing issues

```
+    if (getpid() == ctdbd_pid) tevent_before_wait(epoll_ev->ev);
     ret = epoll_wait(epoll_ev->epoll_fd, events, MAXEVENTS, timeout);
+    if (getpid() == ctdbd_pid) tevent_after_wait(epoll_ev->ev);
```

```
void tevent_before_wait(struct event_context *ev)
{
    diff = tevent_timeval_until(&tevent_after_wait_ts, &now);
    if (diff.tv_sec > 3) {
        tevent_debug(ev, TEVENT_DEBUG_ERROR,  __location__
              " Handling event took %d seconds!", (int) diff.tv_sec);
    }
}
```

# Tevent bug

- Generic framework to instrument tevent

# Tevent bug

- Generic framework to instrument tevent

```
+    tevent_trace_point_callback(epoll_ev->ev, TEVENT_TRACE_BEFORE_WAIT);
     ret = epoll_wait(epoll_ev->epoll_fd, events, MAXEVENTS, timeout);
+    tevent_trace_point_callback(epoll_ev->ev, TEVENT_TRACE_AFTER_WAIT);
```

# Tevent bug

- Generic framework to instrument tevent

```
+    tevent_trace_point_callback(epoll_ev->ev, TEVENT_TRACE_BEFORE_WAIT);
     ret = epoll_wait(epoll_ev->epoll_fd, events, MAXEVENTS, timeout);
+    tevent_trace_point_callback(epoll_ev->ev, TEVENT_TRACE_AFTER_WAIT);
```

```
/**
 * Register a callback to be called at certain trace points
 *
 * @param[in] ev              Event context
 * @param[in] cb              Trace callback
 * @param[in] private_data    Data to be passed to callback
 *
 * @note The callback will be called at trace points defined by
 * tevent_trace_point.  Call with NULL to reset.
 */
void tevent_set_trace_callback(struct tevent_context *ev,
                               tevent_trace_callback_t cb,
                               void *private_data);
```

# Tevent bug

**Problem**

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

- CTDB uses all types of events - fd, timer, immediate, signal

# Tevent bug

## Problem

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

- CTDB uses all types of events - fd, timer, immediate, signal
- Which type of event is taking too long?

# Tevent bug

**Problem**

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

- CTDB uses all types of events - fd, timer, immediate, signal
- Which type of event is taking too long?
  - FD events – Socket handling?

# Tevent bug

- Socket handling
  - Read 1 packet per FD event

# Tevent bug

- Socket handling improvement to reduce recv-Q
  - Read all available data, process packets using immediate events

# Tevent bug

- Socket handling improvement to reduce recv-Q
  - Read all available data, process packets using immediate events

```
static int epoll_event_loop_once(struct tevent_context *ev)
{

    if (ev->immediate_events && tevent_common_loop_immediate(ev)) {
            return 0;
    }

    tval = tevent_common_loop_timer_delay(ev);
    if (tevent_timeval_is_zero(&tval)) {
            return 0;
    }

    return epoll_event_loop(epoll_ev, &tval);
}
```

# Tevent bug

- Socket handling improvement to reduce recv-Q
  - Read all available data, process packets using immediate events
  - Immediate events lead to unfair scheduling across FDs

```
static int epoll_event_loop_once(struct tevent_context *ev)
{

    if (ev->immediate_events && tevent_common_loop_immediate(ev)) {
            return 0;
    }

    tval = tevent_common_loop_timer_delay(ev);
    if (tevent_timeval_is_zero(&tval)) {
            return 0;
    }

    return epoll_event_loop(epoll_ev, &tval);
}
```

# Tevent bug

## Problem

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

- CTDB uses all types of events - fd, timer, immediate, signal
- Which type of event is taking too long?

# Tevent bug

## Problem

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

- CTDB uses all types of events - fd, timer, immediate, signal
- Which type of event is taking too long?
  - FD events – problem resolved using fixed size buffers

# Tevent bug

## Problem

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

- CTDB uses all types of events - fd, timer, immediate, signal
- Which type of event is taking too long?
  - FD events – problem resolved using fixed size buffers
- Need more information ...

# Tevent bug

- Instrument tevent

# Tevent bug

- Instrument tevent
  - Add more trace points

```
+ /* trace point just before calling fd handler function. */
+ TEVENT_TRACE_BEFORE_FD_HANDLER,
+ /* trace point just after calling fd handler function. */
+ TEVENT_TRACE_AFTER_FD_HANDLER,
+
+ /* trace point just before calling timed event handler function. */
+ TEVENT_TRACE_BEFORE_TIMED_HANDLER,
+ /* trace point just after calling timed event handler function. */
+ TEVENT_TRACE_AFTER_TIMED_HANDLER,
+
+ /* trace point just before calling immediate event handler function. */
+ TEVENT_TRACE_BEFORE_IMMEDIATE_HANDLER,
+ /* trace point just after calling immediate event handler function. */
+ TEVENT_TRACE_AFTER_IMMEDIATE_HANDLER,
```

# Tevent bug

- Instrument tevent
  - Add more trace points
  - Count individual types of events

# Tevent bug

- Instrument tevent
  - Add more trace points
  - Count individual types of events

```
2014/02/11 15:50:11 : Handling event took 4 seconds - Got 1 FD events, 12 timed events!
2014/02/11 15:52:18 : Handling timed event took 3 seconds!
2014/02/11 15:53:22 : Handling timed event took 4 seconds!
2014/02/11 15:53:23 : Handling event took 4 seconds - Got 0 FD events, 8 timed events!
2014/02/11 15:54:26 : Handling FD event took 3 seconds!
2014/02/11 15:54:27 : Handling event took 4 seconds - Got 1 FD events, 12 timed events!
2014/02/11 15:55:35 : Handling timed event took 3 seconds!
2014/02/11 15:55:35 : Handling event took 4 seconds - Got 1 FD events, 11 timed events!
2014/02/11 15:58:47 : Handling timed event took 3 seconds!
2014/02/11 15:58:47 : Handling event took 4 seconds - Got 1 FD events, 9 timed events!
2014/02/11 16:00:51 : Handling FD event took 3 seconds!
2014/02/11 16:00:53 : Handling event took 5 seconds - Got 1 FD events, 8 timed events!
2014/02/11 16:02:59 : Handling timed event took 5 seconds!
2014/02/11 16:03:00 : Handling event took 5 seconds - Got 1 FD events, 10 timed events!
2014/02/11 16:05:06 : Handling timed event took 5 seconds!
2014/02/11 16:05:07 : Handling event took 5 seconds - Got 1 FD events, 10 timed events!
2014/02/11 16:07:12 : Handling timed event took 3 seconds!
2014/02/11 16:07:12 : Handling event took 4 seconds - Got 1 FD events, 10 timed events!
2014/02/11 16:09:22 : Handling timed event took 3 seconds!
2014/02/11 16:09:22 : Handling event took 4 seconds - Got 1 FD events, 11 timed events!
```

# Tevent bug

- Instrument tevent
  - Add more trace points
  - Count individual types of events

```
2014/02/11 15:50:11 : Handling event took 4 seconds - Got 1 FD events, 12 timed events!
2014/02/11 15:52:18 : Handling timed event took 3 seconds!
2014/02/11 15:53:22 : Handling timed event took 4 seconds!
2014/02/11 15:53:23 : Handling event took 4 seconds - Got 0 FD events, 8 timed events!
2014/02/11 15:54:26 : Handling FD event took 3 seconds!
2014/02/11 15:54:27 : Handling event took 4 seconds - Got 1 FD events, 12 timed events!
2014/02/11 15:55:35 : Handling timed event took 3 seconds!
2014/02/11 15:55:35 : Handling event took 4 seconds - Got 1 FD events, 11 timed events!
2014/02/11 15:58:47 : Handling timed event took 3 seconds!
2014/02/11 15:58:47 : Handling event took 4 seconds - Got 1 FD events, 9 timed events!
2014/02/11 16:00:51 : Handling FD event took 3 seconds!
2014/02/11 16:00:53 : Handling event took 5 seconds - Got 1 FD events, 8 timed events!
2014/02/11 16:02:59 : Handling timed event took 5 seconds!
2014/02/11 16:03:00 : Handling event took 5 seconds - Got 1 FD events, 10 timed events!
2014/02/11 16:05:06 : Handling timed event took 5 seconds!
2014/02/11 16:05:07 : Handling event took 5 seconds - Got 1 FD events, 10 timed events!
2014/02/11 16:07:12 : Handling timed event took 3 seconds!
2014/02/11 16:07:12 : Handling event took 4 seconds - Got 1 FD events, 10 timed events!
2014/02/11 16:09:22 : Handling timed event took 3 seconds!
2014/02/11 16:09:22 : Handling event took 4 seconds - Got 1 FD events, 11 timed events!
```

- Lots of timer events.

# Tevent bug

- Instrument tevent some more

# Tevent bug

- Instrument tevent some more
  - CTDB uses mostly old style tevent function names *event_add_fd*, *event_add_timed*

```
#ifdef TEVENT_COMPAT_DEFINES

#define event_add_fd(ev, mem_ctx, fd, flags, handler, private_data) \
        tevent_add_fd(ev, mem_ctx, fd, flags, handler, private_data)

#define event_add_timed(ev, mem_ctx, next_event, handler, private_data) \
        tevent_add_timer(ev, mem_ctx, next_event, handler, private_data)

#endif /* TEVENT_COMPAT_DEFINES */
```

# Tevent bug

- Instrument tevent some more
  - CTDB uses mostly old style tevent function names *event_add_fd*, *event_add_timed*
  - Create wrappers for these functions

```
#undef event_add_fd
#undef event_add_timed

#define event_add_fd(ev, mem_ctx, fd, flags, handler, private_data) \
_event_add_fd(ev, mem_ctx, fd, flags, handler, private_data, #handler)

#define event_add_timed(ev, mem_ctx, next, handler, private_data) \
_event_add_timed(ev, mem_ctx, next, handler, private_data, #handler)

void ctdb_event_stack_clear(void);
void ctdb_event_stack_dump(void);
```

# Tevent bug

- Instrument tevent some more
  - CTDB uses mostly old style tevent function names *event_add_fd*, *event_add_timed*
  - Create wrappers for these functions

```
2014/02/13 15:09:04: Handling event took 9 seconds - Got 1 FD events, 9 timed events, 0 immediate events!
2014/02/13 15:09:04: event_stack: TIMER ctdb_check_for_dead_nodes (1) - 0.000069 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_vacuum_event (1) - 0.033250 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_check_health (1) - 0.006896 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_vacuum_event (1) - 0.007691 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_vacuum_event (1) - 0.250795 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_time_tick (1) - 0.000005 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_ltdb_seqnum_check (1) - 0.000005 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_statistics_update (1) - 0.000130 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_vacuum_event (1) - 7.964969 seconds
2014/02/13 15:09:04: event_stack: FD queue_io_handler (1) - 0.811996 seconds

2014/02/13 15:10:11: Handling event took 6 seconds - Got 0 FD events, 12 timed events, 0 immediate events
2014/02/13 15:10:11: event_stack: TIMER ctdb_check_for_dead_nodes (1) - 0.000067 seconds
2014/02/13 15:10:11: event_stack: TIMER ctdb_check_health (1) - 0.003055 seconds
2014/02/13 15:10:11: event_stack: TIMER ctdb_time_tick (1) - 0.000005 seconds
2014/02/13 15:10:11: event_stack: TIMER ctdb_ltdb_seqnum_check (1) - 0.000005 seconds
2014/02/13 15:10:11: event_stack: TIMER ctdb_statistics_update (1) - 0.000182 seconds
2014/02/13 15:10:11: event_stack: TIMER ctdb_vacuum_event (7) - 6.464652 seconds
```

# Tevent bug

- Instrument tevent some more
  - CTDB uses mostly old style tevent function names *event_add_fd*, *event_add_timed*
  - Create wrappers for these functions

```
2014/02/13 15:09:04: Handling event took 9 seconds - Got 1 FD events, 9 timed events, 0 immediate events!
2014/02/13 15:09:04: event_stack: TIMER ctdb_check_for_dead_nodes (1) - 0.000069 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_vacuum_event (1) - 0.033250 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_check_health (1) - 0.006896 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_vacuum_event (1) - 0.007691 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_vacuum_event (1) - 0.250795 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_time_tick (1) - 0.000005 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_ltdb_seqnum_check (1) - 0.000005 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_statistics_update (1) - 0.000130 seconds
2014/02/13 15:09:04: event_stack: TIMER ctdb_vacuum_event (1) - 7.964969 seconds
2014/02/13 15:09:04: event_stack: FD queue_io_handler (1) - 0.811996 seconds

2014/02/13 15:10:11: Handling event took 6 seconds - Got 0 FD events, 12 timed events, 0 immediate events
2014/02/13 15:10:11: event_stack: TIMER ctdb_check_for_dead_nodes (1) - 0.000067 seconds
2014/02/13 15:10:11: event_stack: TIMER ctdb_check_health (1) - 0.003055 seconds
2014/02/13 15:10:11: event_stack: TIMER ctdb_time_tick (1) - 0.000005 seconds
2014/02/13 15:10:11: event_stack: TIMER ctdb_ltdb_seqnum_check (1) - 0.000005 seconds
2014/02/13 15:10:11: event_stack: TIMER ctdb_statistics_update (1) - 0.000182 seconds
2014/02/13 15:10:11: event_stack: TIMER ctdb_vacuum_event (7) - 6.464652 seconds
```

- ctdb_vacuum_event is the culprit!

# Tevent bug

## Problem

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

- CTDB uses all types of events - fd, timer, immediate, signal
- Which type of event is taking too long?
  - FD events – problem resolved using fixed size buffers
  - timer events – ctdb_vacuum_event

# Tevent bug

## Problem

ctdbd: ./lib/tevent/tevent_util.c:110 Handling event took 129 seconds!

- CTDB uses all types of events - fd, timer, immediate, signal
- Which type of event is taking too long?
  - FD events – problem resolved using fixed size buffers
  - timer events – ctdb_vacuum_event
- Does it explain?

# Tevent bug

- ctdb_vacuum_event

# Tevent bug

- ctdb_vacuum_event
  - Runs every 10 seconds

# Tevent bug

- ctdb_vacuum_event
  - Runs every 10 seconds for each volatile database

# Tevent bug

- ctdb_vacuum_event
  - Runs every 10 seconds for each volatile database
  - Create a vacuuming child process with fork()

# Tevent bug

- ctdb_vacuum_event
  - Runs every 10 seconds for each volatile database
  - Create a vacuuming child process with fork()

Time (in $\mu$s) required to create a child process

| Memory | 0M | 10M | 100M |
|---|---|---|---|
| **fork** | $41 \pm 3$ | $151 \pm 10$ | $1075 \pm 61$ |

# Tevent bug

- ctdb_vacuum_event
  - Runs every 10 seconds for each volatile database
  - Create a vacuuming child process with fork()
  - On a busy system with large tdb databases (locking, brlock), this can be quite expensive

# Tevent bug

- ctdb_vacuum_event
  - Runs every 10 seconds for each volatile database
  - Create a vacuuming child process with fork()
  - On a busy system with large tdb databases (locking, brlock), this can be quite expensive
  - If multiple vacuuming events trigger at the same time, ...

# Tevent bug

- ctdb_vacuum_event
  - Runs every 10 seconds for each volatile database
  - Create a vacuuming child process with fork()
  - On a busy system with large tdb databases (locking, brlock), this can be quite expensive
  - If multiple vacuuming events trigger at the same time, ...
    - Controls to CTDB daemon can time out

# Tevent bug

- ctdb_vacuum_event
  - Runs every 10 seconds for each volatile database
  - Create a vacuuming child process with fork()
  - On a busy system with large tdb databases (locking, brlock), this can be quite expensive
  - If multiple vacuuming events trigger at the same time, ...
    - Controls to CTDB daemon can time out
- Solutions

# Tevent bug

- ctdb_vacuum_event
  - Runs every 10 seconds for each volatile database
  - Create a vacuuming child process with fork()
  - On a busy system with large tdb databases (locking, brlock), this can be quite expensive
  - If multiple vacuuming events trigger at the same time, ...
    - Controls to CTDB daemon can time out
- Solutions
  - Stagger vacuuming child processes?

# Tevent bug

- ctdb_vacuum_event
  - Runs every 10 seconds for each volatile database
  - Create a vacuuming child process with fork()
  - On a busy system with large tdb databases (locking, brlock), this can be quite expensive
  - If multiple vacuuming events trigger at the same time, ...
    - Controls to CTDB daemon can time out
- Solutions
  - Stagger vacuuming child processes?
  - Create a long running vacuuming daemon?

# Tevent bug

- ctdb_vacuum_event
  - Runs every 10 seconds for each volatile database
  - Create a vacuuming child process with fork()
  - On a busy system with large tdb databases (locking, brlock), this can be quite expensive
  - If multiple vacuuming events trigger at the same time, ...
    - Controls to CTDB daemon can time out
- Solutions
  - Stagger vacuuming child processes?
  - Create a long running vacuuming daemon?
- Vacuuming has other problems too ...

# Vacuuming performance

**Problem**

ctdbd: Vacuuming child process timed out for db locking.tdb

- Timer to track run-away vacuuming (120 seconds)

# Vacuuming performance

**Problem**

ctdbd: Vacuuming child process timed out for db locking.tdb

- Timer to track run-away vacuuming (120 seconds)
- What is Vacuuming? Why is it important?

# Vacuuming performance

**Problem**

ctdbd: Vacuuming child process timed out for db locking.tdb

- Timer to track run-away vacuuming (120 seconds)
- What is Vacuuming? Why is it important?
- When Samba is done with a record, it deletes it from TDB.

# Vacuuming performance

**Problem**

ctdbd: Vacuuming child process timed out for db locking.tdb

- Timer to track run-away vacuuming (120 seconds)
- What is Vacuuming? Why is it important?
- When Samba is done with a record, it deletes it from TDB. (e.g. When a file is closed, locking.tdb record is deleted)

# Vacuuming performance

**Problem**

ctdbd: Vacuuming child process timed out for db locking.tdb

- Timer to track run-away vacuuming (120 seconds)

- What is Vacuuming? Why is it important?
- When Samba is done with a record, it deletes it from TDB. (e.g. When a file is closed, locking.tdb record is deleted)
- With CTDB, the record cannot be deleted immediately.

# Vacuuming performance

**Problem**

ctdbd: Vacuuming child process timed out for db locking.tdb

- Timer to track run-away vacuuming (120 seconds)

- What is Vacuuming? Why is it important?
- When Samba is done with a record, it deletes it from TDB. (e.g. When a file is closed, locking.tdb record is deleted)
- With CTDB, the record cannot be deleted immediately.
  - Instead, **mark** the record as deleted (empty data),

# Vacuuming performance

**Problem**

ctdbd: Vacuuming child process timed out for db locking.tdb

- Timer to track run-away vacuuming (120 seconds)

- What is Vacuuming? Why is it important?
- When Samba is done with a record, it deletes it from TDB. (e.g. When a file is closed, locking.tdb record is deleted)
- With CTDB, the record cannot be deleted immediately.
  - Instead, **mark** the record as deleted (empty data),
  - and **delete** from all the nodes (vacuuming).

# Vacuuming performance

**Problem**

ctdbd: Vacuuming child process timed out for db locking.tdb

- Timer to track run-away vacuuming (120 seconds)

- What is Vacuuming? Why is it important?
- When Samba is done with a record, it deletes it from TDB. (e.g. When a file is closed, locking.tdb record is deleted)
- With CTDB, the record cannot be deleted immediately.
  - Instead, **mark** the record as deleted (empty data),
  - and **delete** from all the nodes (vacuuming).
- Why not delete the record from all nodes immediately?

# Vacuuming performance

**Problem**

ctdbd: Vacuuming child process timed out for db locking.tdb

- Timer to track run-away vacuuming (120 seconds)

- What is Vacuuming? Why is it important?
- When Samba is done with a record, it deletes it from TDB. (e.g. When a file is closed, locking.tdb record is deleted)
- With CTDB, the record cannot be deleted immediately.
  - Instead, **mark** the record as deleted (empty data),
  - and **delete** from all the nodes (vacuuming).
- Why not delete the record from all nodes immediately?
  - Performance

# Vacuuming performance

**Problem**

ctdbd: Vacuuming child process timed out for db locking.tdb

- Timer to track run-away vacuuming (120 seconds)

- What is Vacuuming? Why is it important?
- When Samba is done with a record, it deletes it from TDB. (e.g. When a file is closed, locking.tdb record is deleted)
- With CTDB, the record cannot be deleted immediately.
  - Instead, **mark** the record as deleted (empty data),
  - and **delete** from all the nodes (vacuuming).
- Why not delete the record from all nodes immediately?
  - Performance
  - What happens if deleting fails on a remote node?

# Vacuuming performance

- Vacuuming Process
  1. On *dmaster* migrate empty record to *lmaster*
  2. On *lmaster*, write empty record to the other nodes
  3. On *lmaster*, delete empty record from the other nodes
  4. On *lmaster*, delete record locally

# Vacuuming performance

- Vacuuming Process
  1. On *dmaster* migrate empty record to *lmaster*
  2. On *lmaster*, write empty record to the other nodes
  3. On *lmaster*, delete empty record from the other nodes
  4. On *lmaster*, delete record locally
- This process is repeated for each and every deleted record

# Vacuuming performance

- Vacuuming Process
  1. On *dmaster* migrate empty record to *lmaster*
  2. On *lmaster*, write empty record to the other nodes
  3. On *lmaster*, delete empty record from the other nodes
  4. On *lmaster*, delete record locally
- This process is repeated for each and every deleted record
- Need to handle data corruption if database recovery happens!

# Vacuuming performance

- Vacuuming Process
  1. On *dmaster* migrate empty record to *lmaster*
  2. On *lmaster*, write empty record to the other nodes
  3. On *lmaster*, delete empty record from the other nodes
  4. On *lmaster*, delete record locally
- This process is repeated for each and every deleted record
- Need to handle data corruption if database recovery happens!
- At every stage, check if the record is still empty

# Vacuuming performance

- Vacuuming Process
  1. On *dmaster* migrate empty record to *lmaster*
  2. On *lmaster*, write empty record to the other nodes
  3. On *lmaster*, delete empty record from the other nodes
  4. On *lmaster*, delete record locally
- This process is repeated for each and every deleted record
- Need to handle data corruption if database recovery happens!
- At every stage, check if the record is still empty
  - If an operation fails, skip the record

# Vacuuming performance

- Vacuuming Process
  1. On *dmaster* migrate empty record to *lmaster*
  2. On *lmaster*, write empty record to the other nodes
  3. On *lmaster*, delete empty record from the other nodes
  4. On *lmaster*, delete record locally
- This process is repeated for each and every deleted record
- Need to handle data corruption if database recovery happens!
- At every stage, check if the record is still empty
  - If an operation fails, skip the record
- Periodic database traverse to check any skipped records

# Vacuuming performance

- Vacuuming Process
  1. On *dmaster* migrate empty record to *lmaster*
  2. On *lmaster*, write empty record to the other nodes
  3. On *lmaster*, delete empty record from the other nodes
  4. On *lmaster*, delete record locally
- This process is repeated for each and every deleted record
- Need to handle data corruption if database recovery happens!
- At every stage, check if the record is still empty
  - If an operation fails, skip the record
- Periodic database traverse to check any skipped records
- Vacuuming can get in the way of regular record processing

# Vacuuming performance

- Improvements

# Vacuuming performance

- Improvements
  - Use tdb_parse_record() instead of tdb_fetch()

# Vacuuming performance

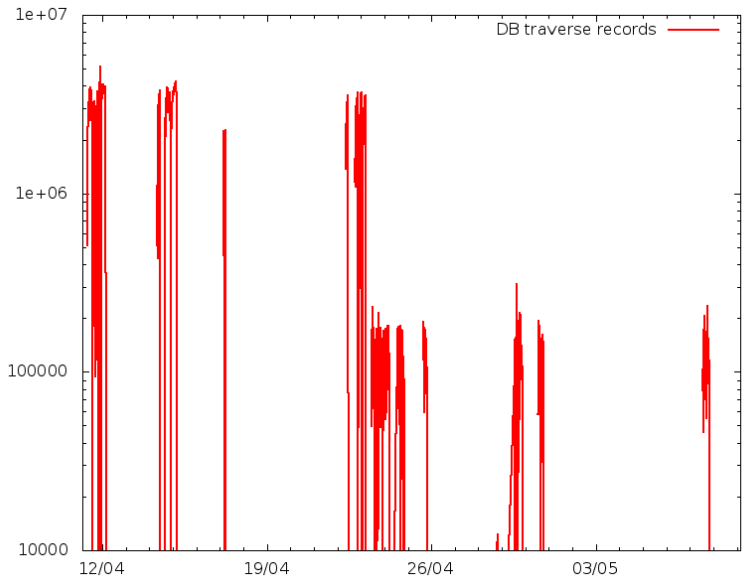
- Improvements
  - Use tdb_parse_record() instead of tdb_fetch()
  - Use non-blocking lock when traversing delete queue

# Vacuuming performance

- Improvements
  - Use tdb_parse_record() instead of tdb_fetch()
  - Use non-blocking lock when traversing delete queue
  - Improve VACUUM FETCH processing in recovery daemon

# Vacuuming performance

- Improvements
  - Use tdb_parse_record() instead of tdb_fetch()
  - Use non-blocking lock when traversing delete queue
  - Improve VACUUM FETCH processing in recovery daemon



# Going forward

# The future?

- Split monolithic code into separate daemons
  - Logging, IP handling, Services monitoring
- Missing CTDB library – libctdb
  - Require async API
  - Thread-safe
- CTDB Protocol
  - Auto-generated marshalling/unmarshalling code
  - Version tracking
- Scalability – large number of nodes
  - Database recovery
  - Handling record contention
  - Vacuuming
- Pluggable Monitoring and Failover
  - Integration with 3rd party HA

Questions/Comments?