Tracing & Visualization in Barrelfish

Rebecca Isaacs

Barcelona Barrelfish Workshop

September 2010

Trace System Design

- See the behaviour of the live system
  - Scheduling, concurrency, messages
  - Domain-specific events
- Low overhead
  - Try not to perturb the system being traced
- Low level
  - Can trace **anything** from **anywhere**
    - Minimal dependencies on other components
    - Does not rely on IDC, scheduling, memory allocation
  - Support tracing of system start-up

Overview

- Use tracing for:
  - Demos
  - Performance and concurrency debugging
    - How does your new feature work?
    - Does it mess up something else?
- Do not use tracing for:
  - Functional debugging
- This talk
  - Quick run-through of the design and implementation
  - Hopefully persuade you that it’s worth using!

Summary

- Tracing (lib/trace)
  - Instrumented applications write events to the trace buffer using `trace_event(…)`
  - One buffer per core
  - Turned on and off dynamically
- Visualization (Aquarium)
  - GUI that shows an event timeline per core
    - And messages sent between cores
  - Runs on Windows only
    - Code was reused from various older research projects

Events

- 64-bit timestamp + 64-bit body
  - The body can be “raw” or encode the source of the event (the subsystem), the event type and an argument
  - Constants for trace subsystems and events are defined in include/trace/trace.h
- Example:

```
TRACE_SUBSYS_MEMSERV = 0xA000
TRACE_EVENT_ALLOC = 0x0001
```

memserv.c/mem_allocate_handler() calls

```
trace_event(TRACE_SUBSYS_MEMSERV,TRACE_EVENT_ALLOC,bits);
```

If `bits`=17, then the event posted is `0xA000000100000011`

Buffers

- Trace buffers contain an array of events, plus control fields
  - One per core, initialized when the core is booted
- Buffers are fixed size
  - Tracing will stop when the buffer fills up, or when the specified trace duration is exceeded

Data structures

- Master data structure for control
  - Shared, largely read-only
- Each core's trace buffer has private head and tail pointers
  - Atomic updates to write events into the buffer
- Implementation is architecture specific
  - On x86_64 the virtual address of the core-local buffer is computed from the core id

Control

- Tracing is turned on and off globally via trigger events
  - On x86, the flag `master->running` is set or unset
- Specify the trigger events by calling

```
trace_control(START_TRIGGER_EVENT,
              STOP_TRIGGER_EVENT,
              duration);
```

  - Trigger events can be any event type

Producing a trace

1. Clear the buffers from the previous session

   ```
   trace_reset_all();
   ```

2. Specify start and stop events and duration

   ```
   trace_control(START_TRIGGER, STOP_TRIGGER,
                 duration);
   ```

3. Somebody issues the start event

   ```
   trace_event(START_TRIGGER);
   ```

4. Do work… logging happens…
5. Somebody issues the stop event

   ```
   trace_event(STOP_TRIGGER);
   ```

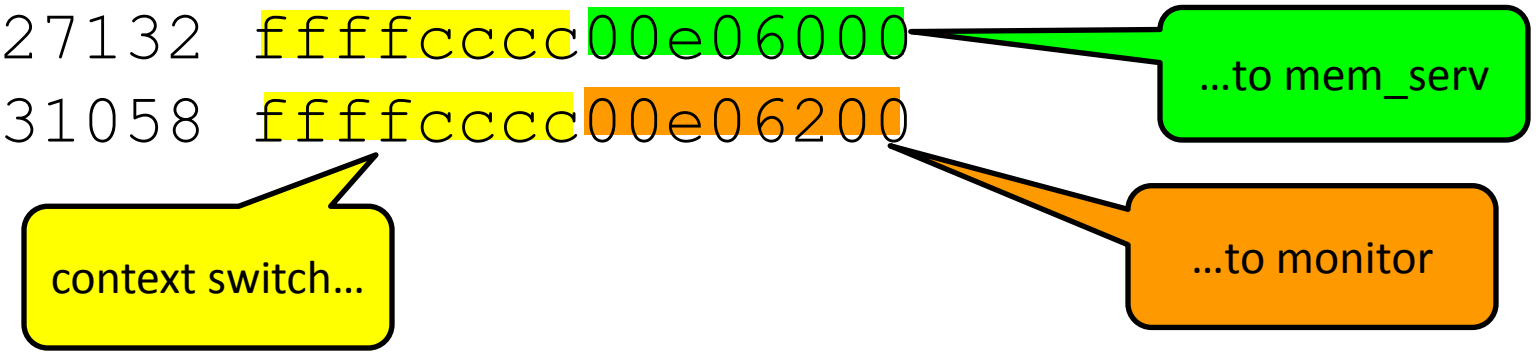
6. Dump the trace

   ```
   trace_dump(…);
   ```

DCB Rundown

- Special events record the list of current DCBs
  - Top bit set in timestamp means rundown event
- Used to parse context switch events
- Implemented (for x86 only) in the kernel function `trace_snapshot()`
- Example:

```
# DCB 0 ffffff0000e06000 mem_serv
# DCB 0 ffffff0000e06200 monitor
0 27132 ffffcccc00e06000
0 31058 ffffcccc00e06200
```



Optimization: global coordination

- On x86_64, notifications are sent on the start and stop trigger events
- `IPI_TRACE_START` (=63) is sent to all cores
  - Each core calls trace_snapshot() to get its DCB rundown
- `IPI_TRACE_COMPLETE` (=64) is sent to a single core
  - Enables a user-space domain to wait for trace completion without polling the master

Miscellaneous

- Tracing is not enabled by default
  - Turn on in Config.hs
- Portability requirements:
  - Core-local atomic update (e.g. CAS, cli/sti) and timestamp functions
  - Some way to share the master across all cores
- Limitations:
  - No security
    - Anyone can read or write the trace buffer
  - No protection
    - Anyone can corrupt the trace buffer
  - Fixed size events and fixed size buffers constrain expressiveness

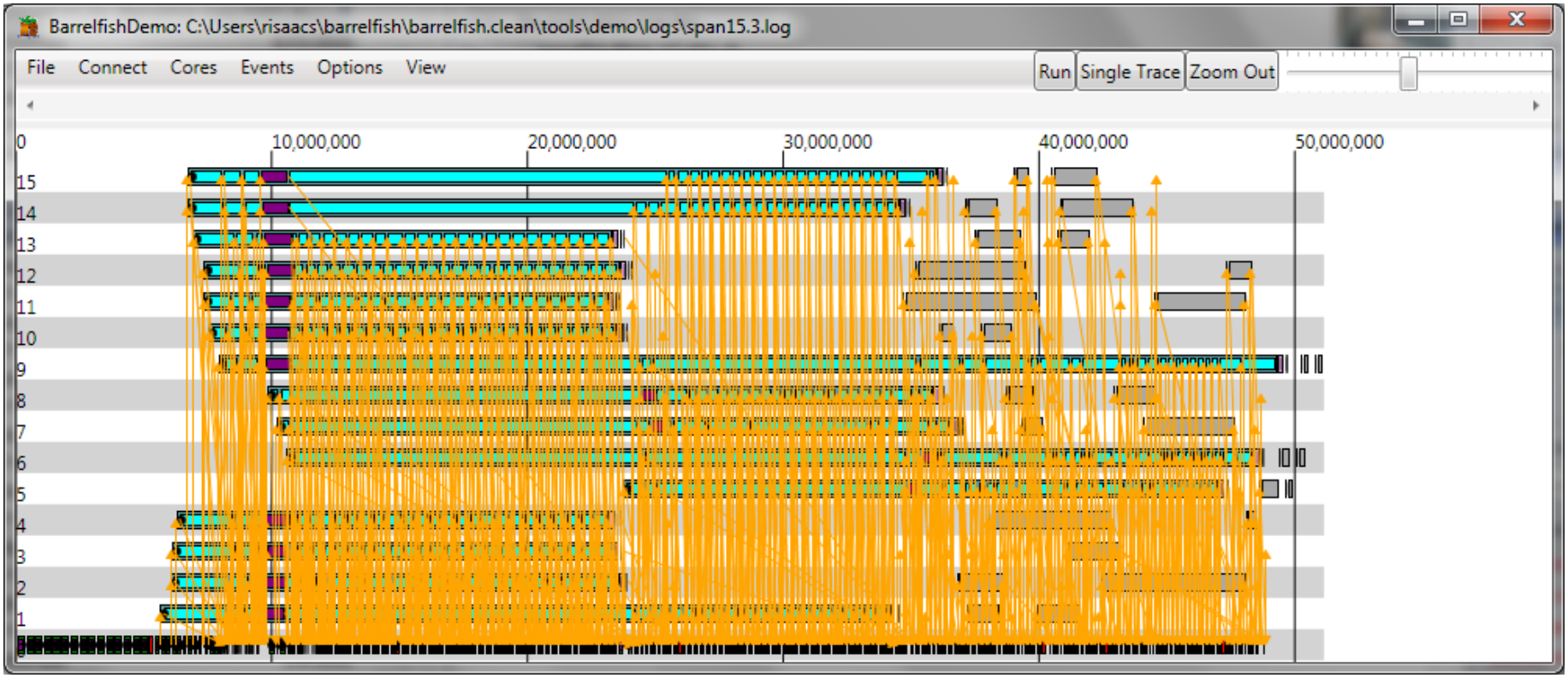
bfscope

- Listens on TCP port 666 for “trace” commands
  - Sets up a tracing session with default trigger events
  - Waits for the trace complete notification
  - Returns the formatted events in the buffers
- Demo tool for x86_64
  - Now defunct?

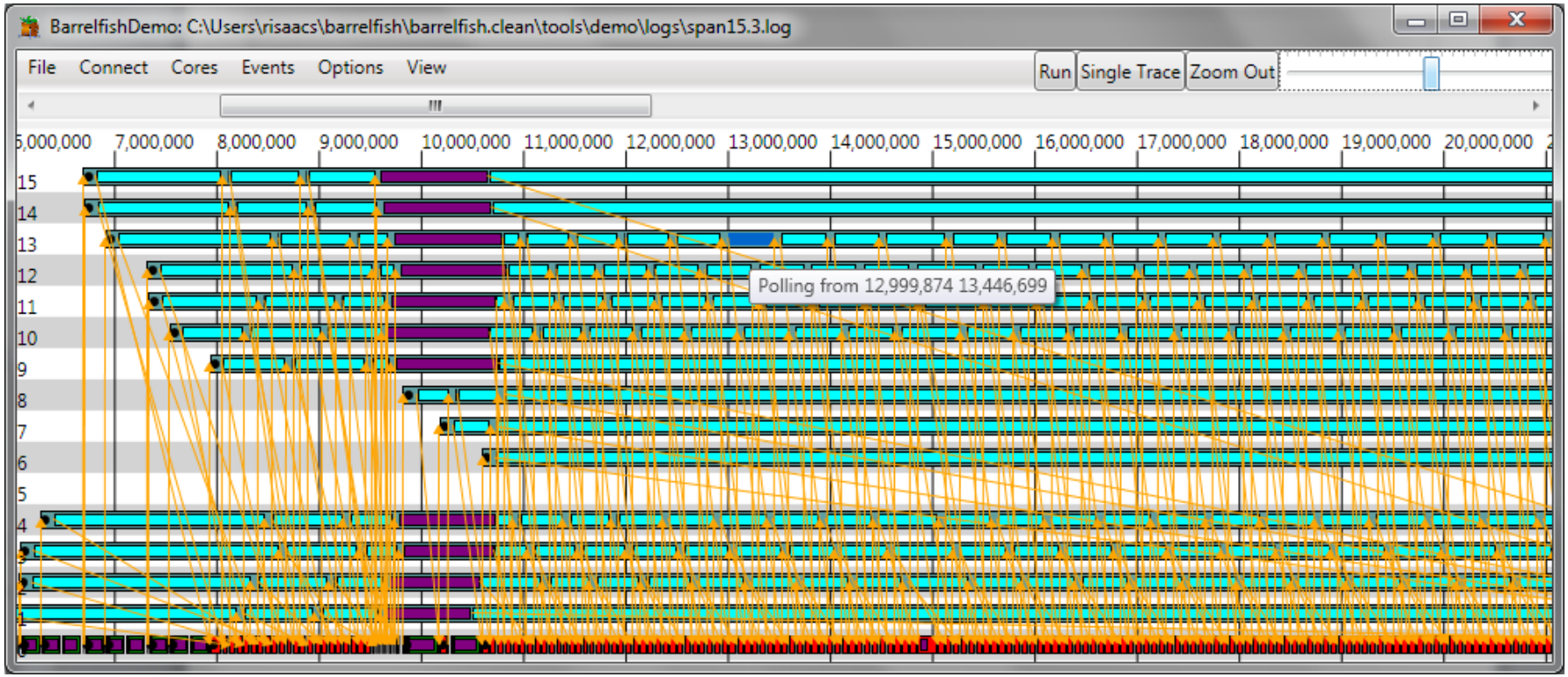
Aquarium

- Can run “live” as a client of bfscope
- Or display previously saved trace files

Example: 16 core spantest



Zoom in, use tooltips



Zoom in to diagnose

To do

- Support a circular trace buffer
- Filter with an event/subsystem mask
  - Currently tracing is either on or off, which can lead to many extraneous events
- Rewrite of Aquarium

Further reading: K42 tracing