Google

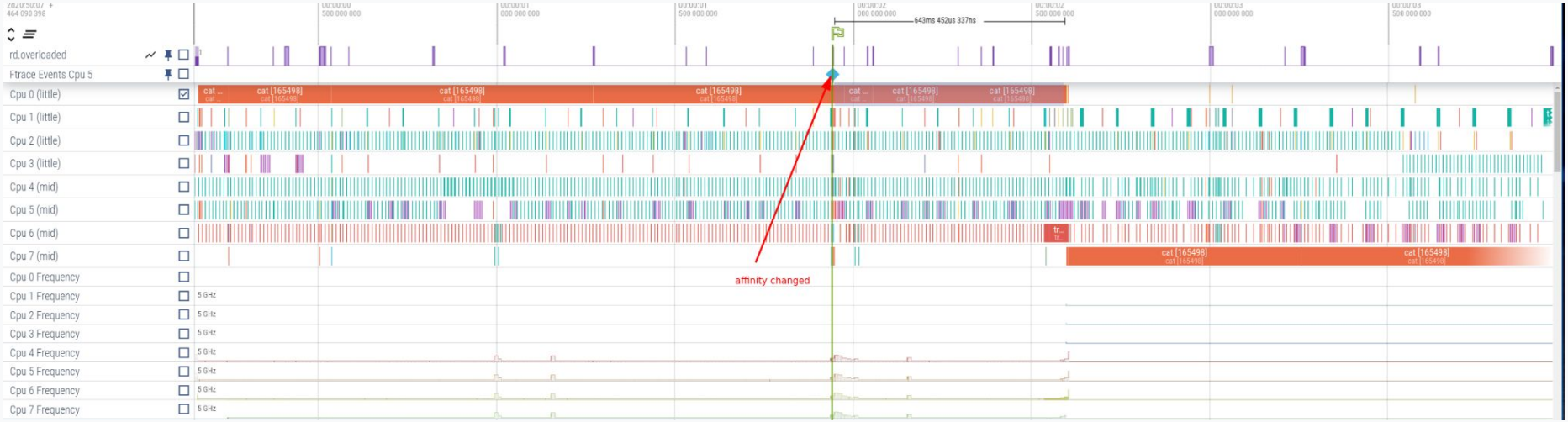
Improve scheduler debuggability

Qais Yousef

Sep 2024

Understanding when things go wrong is hard - wakeup

Understanding when things go wrong is hard - lb



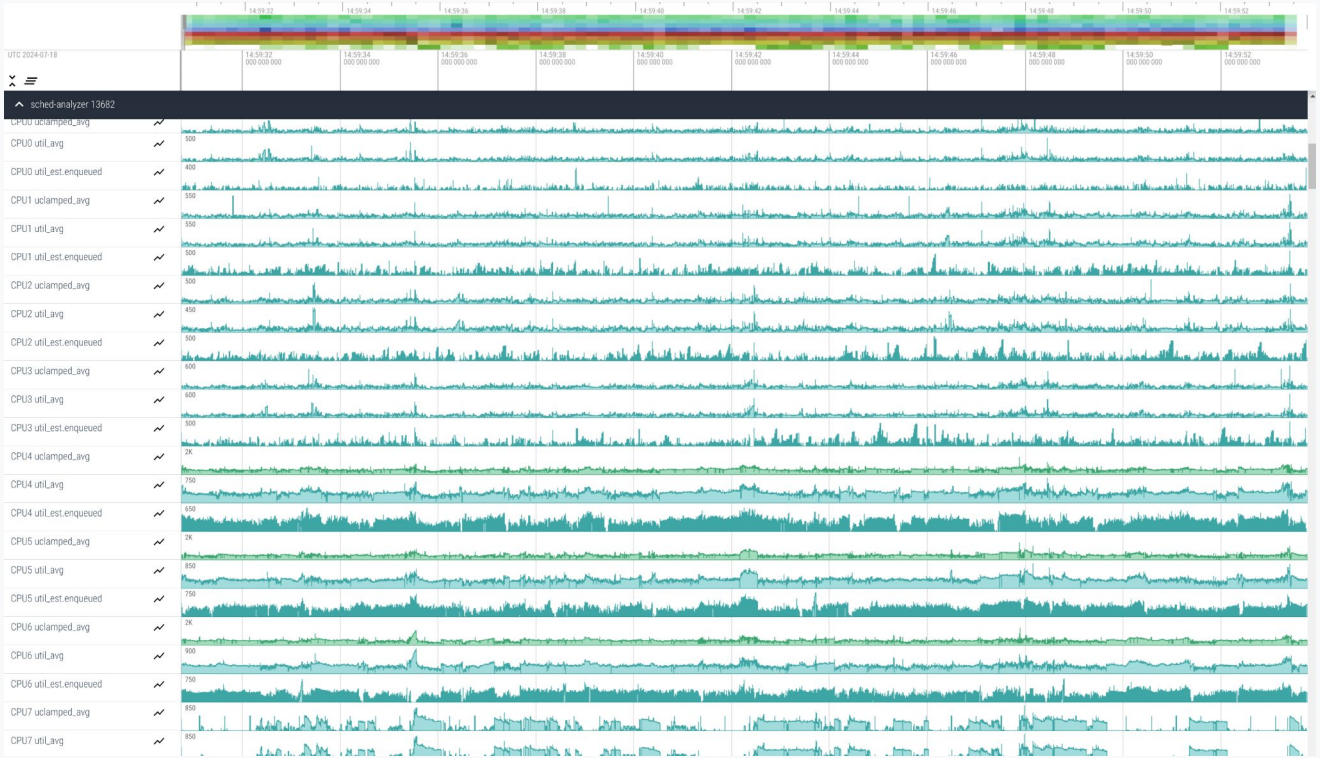
We need to make debugging easier

- One of the major bottlenecks in upstream discussions is getting people on the same page
- No one way to communicate problems/results
- Users (both system admins, app developers and regular users) understanding/expectations is off the mark majority of the time
    - ‘Bad latencies’ is a common phrase that is used to describe different problems but get clumped together as one in many discussions
- New developers struggle to grasp the full aspect of the code as it lacks annotation
- A formally supported annotation of tricky code paths can help alleviate these problems
    - pr_debug(), but without pr_debug()!

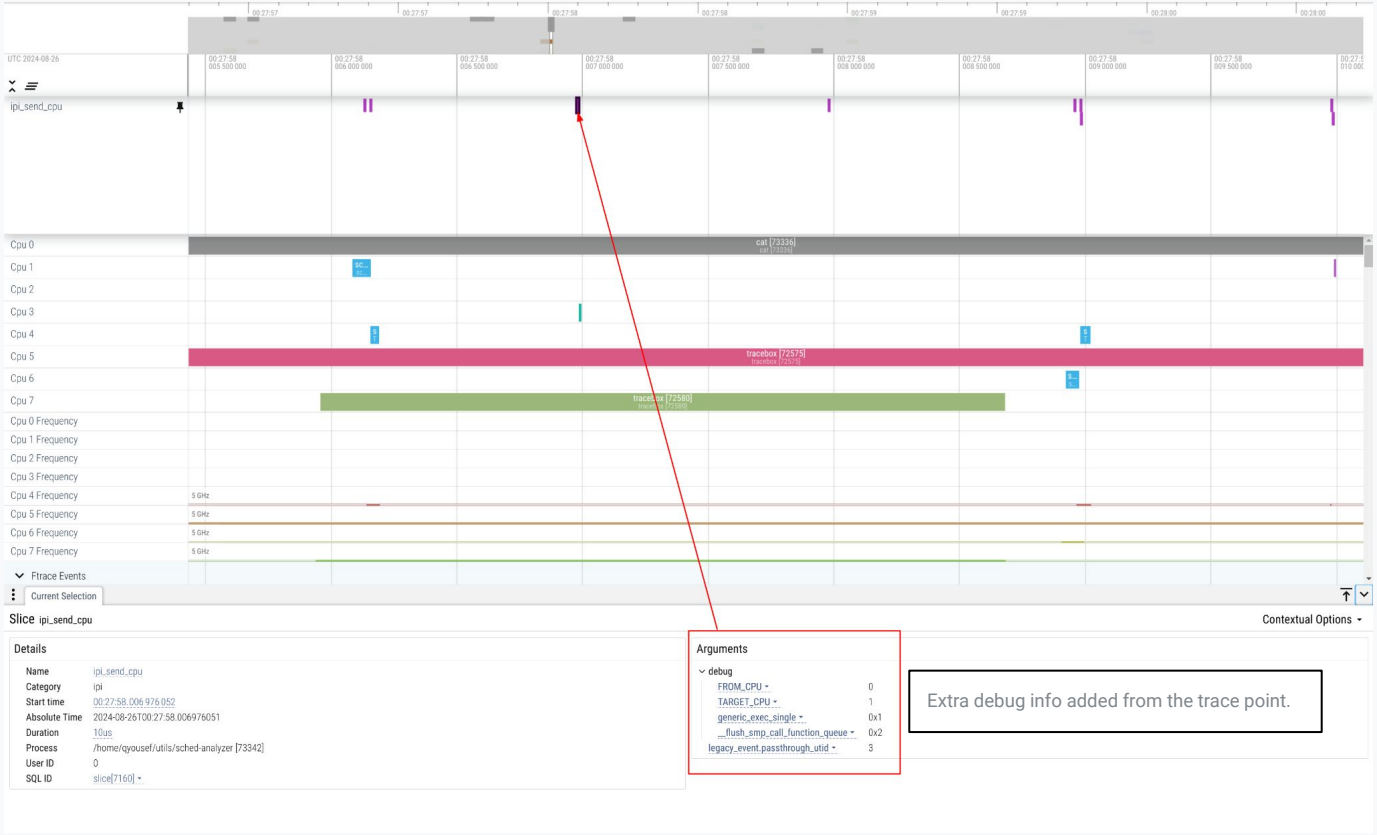
Sched-analyzer is an attempt to formalize the effort

- Extracting internal scheduler details is hard without being able to add new trace events
- Even if we add trace events, text based debug is hard. We need to visualize when a problem happen and then inspect internal details around that problematic point
- sched-analyzer is a glue logic around existing mature technologies
  - Ftrace
  - BPF
  - Perfetto
  - Pandas
- We just connect the dots to produce a single binary (a static version of it can run on any system)
- Ftrace and BPF help to connect to the bare trace points and other points of interest to extract data and emit them as perfetto events
- Perfetto helps to visualize the trace and process it via SQL
- Pandas allows creating powerful post processing to aid with more analysis and auto detection of problems

Visualize PELT signals - for cpus and tasks

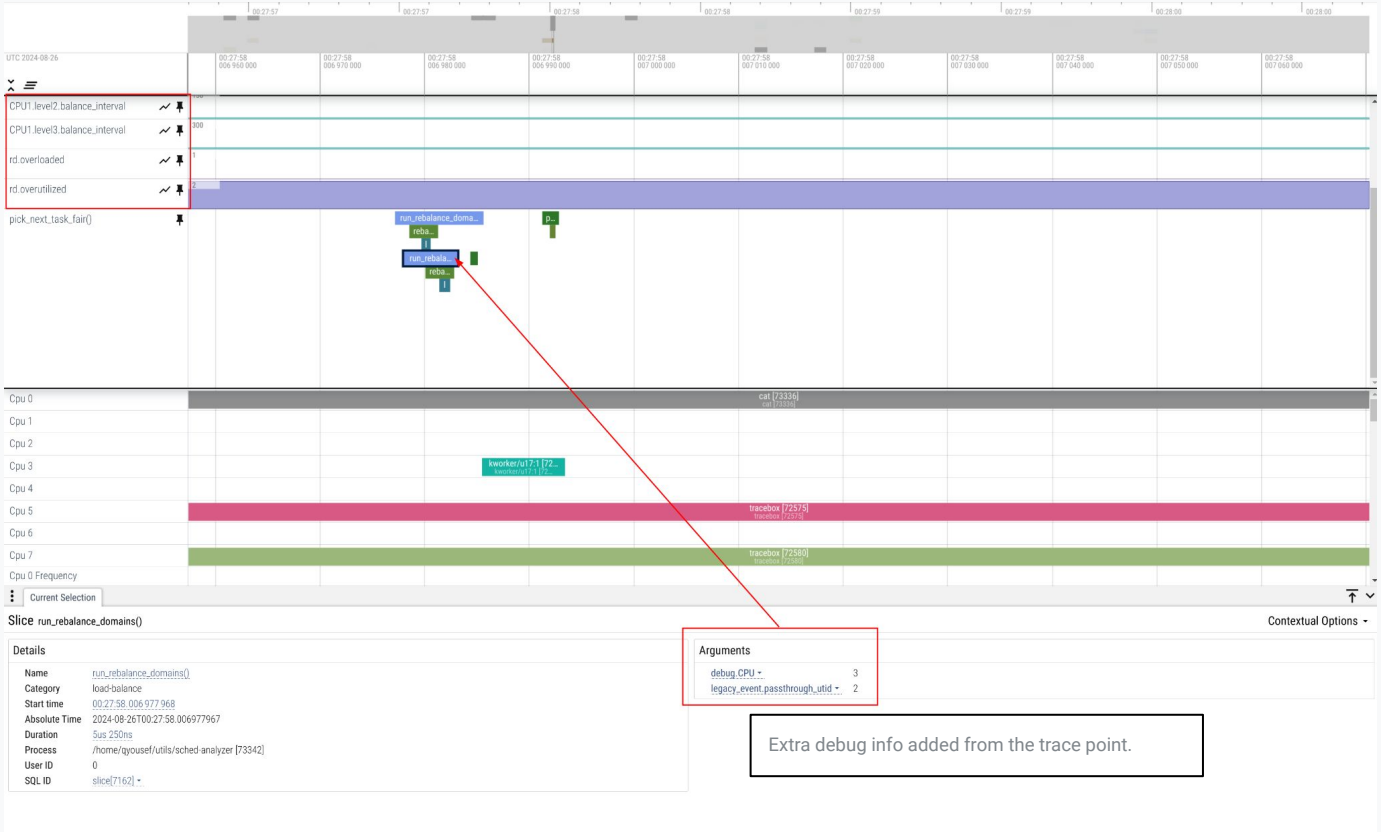


Visualize IPIs



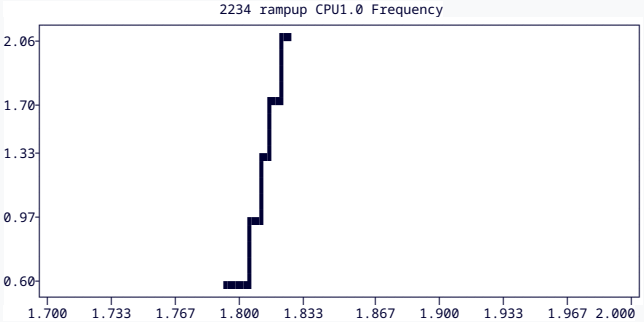
Visualize Load balance

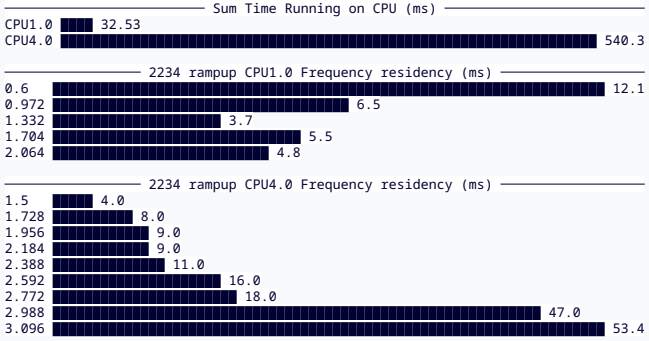
Extra info about overeloaded, overutilized and balance_interval are also captured, we can easily add more

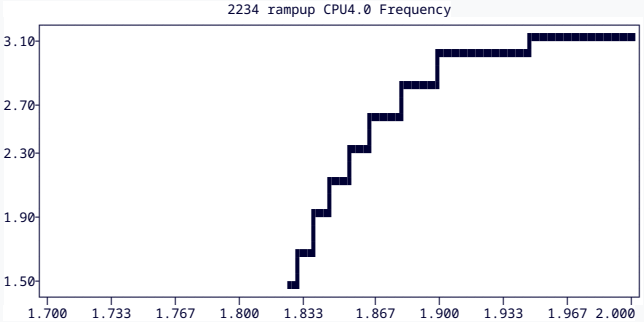


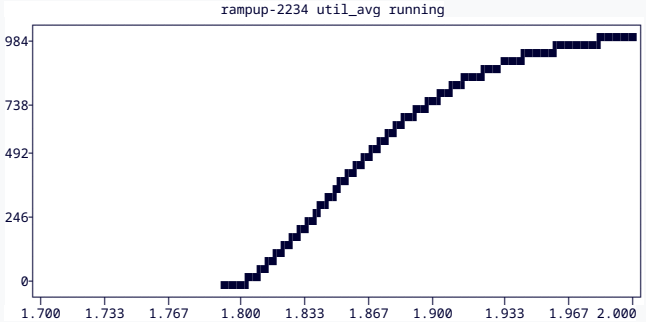
Google

Can produce TUI that is shareable on the list









Current limitation to trace wakeup/load_balance

- Functions are heavily inlined and hard to hook into them reliably across kernels
- Relying on hooking into function entry and exit is very limited
    - We want to extract information from decision taken somewhere inside a function
- We want the information to be useful for end users to better understand their bottlenecks and write better software
- And for developers to help debug issues from the field where it is impossible to reproduce locally

What can we do?

- One idea is to replicate perfetto's (not related to kernel's) TRACE_EVENT() macro
- It takes arbitrary 'key':value pairs and this is what was used to produce current IPI and load_balance tracks

3. Arbitrary number of debug annotations:

```
TRACE_EVENT("category", "Name", "arg", value);
TRACE_EVENT("category", "Name", "arg", value, "arg2", value2);
TRACE_EVENT("category", "Name", "arg", value, "arg2", value2,
            "arg3", value3);
```

DECLARE_TRACE_PATH()/trace_path()

```
#define DECLARE_TRACE_PATH(name)                                        \
        DECLARE_TRACE(name,                                             \
                TP_PROTO(const char *phase,                             \
                         const char *debug_name1, s64 val1,             \
                         const char *debug_name2, s64 val2,             \
                         const char *debug_name3, s64 val3),            \
                TP_ARGS(phase,                                          \
                        debug_name1, val1,                              \
                        debug_name2, val2,                              \
                        debug_name3, val3))

// macro magic

#define trace_path(name, ...)                                           \
        _trace_path(trace_path_, _NARGS(__VA_ARGS__))(name, __func__,
__VA_ARGS__)

DECLARE_TRACE_PATH(select_task_rq_fair);

// fair.c
trace_path(select_task_rq_fair, "wake_flags", wake_flags);
```

We should be able to filter output with SQL/pandas

- Visualization will make it easier to catch point of interest and inspect further
- Writing scripts to find if a problem exists in a trace should be easy in principle
- If the output is too large, we can implement filtering mechanism in sched-analyzer to only emit an event if certain conditions were met

Any other ideas?