Comparison of job resource consumption at different cache states

The following is a test that shows the effects of the Linux filesystem cache being „hot“ or „cold“ (filled with data needed for the job) and the PostgreSQL shared_buffers caches being „hot“ or „cold“ on job run-time and resource usage profiles.

In particular the CPU load, the ratio between system and user CPU and the storage IO load consumption profiles change when caches are cold versus hot.

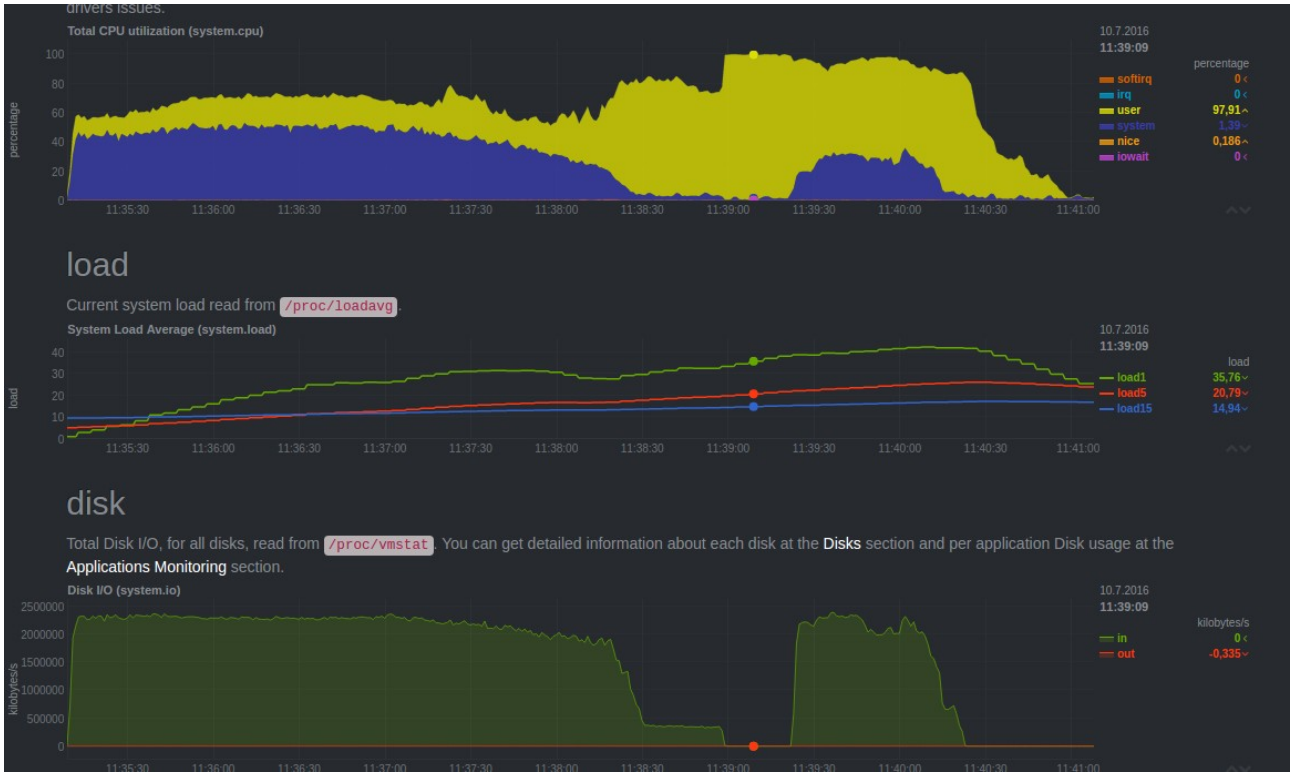
Test Job

The test job used can be found in Git here:

`<GIT-RA>/adr/services/svc_sqlbalancer/examples/rating_ereignis_summary.sql`

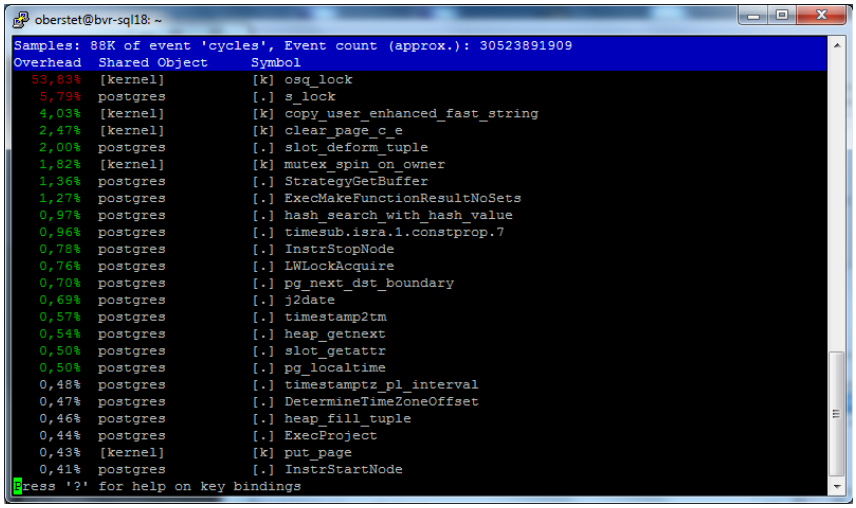
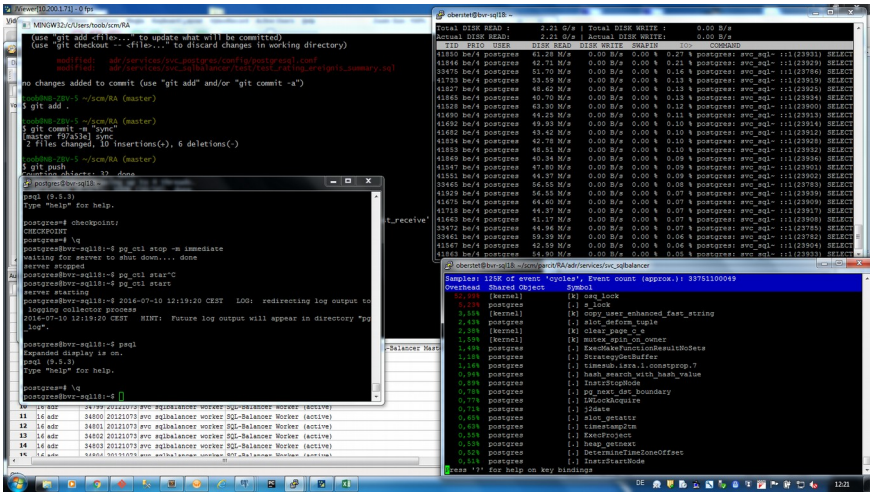
PG-Cold / FS-Cold

Here is the CPU/disk load during a complete run of the test job with PostgreSQL cache cold and Filesystem cache cold:



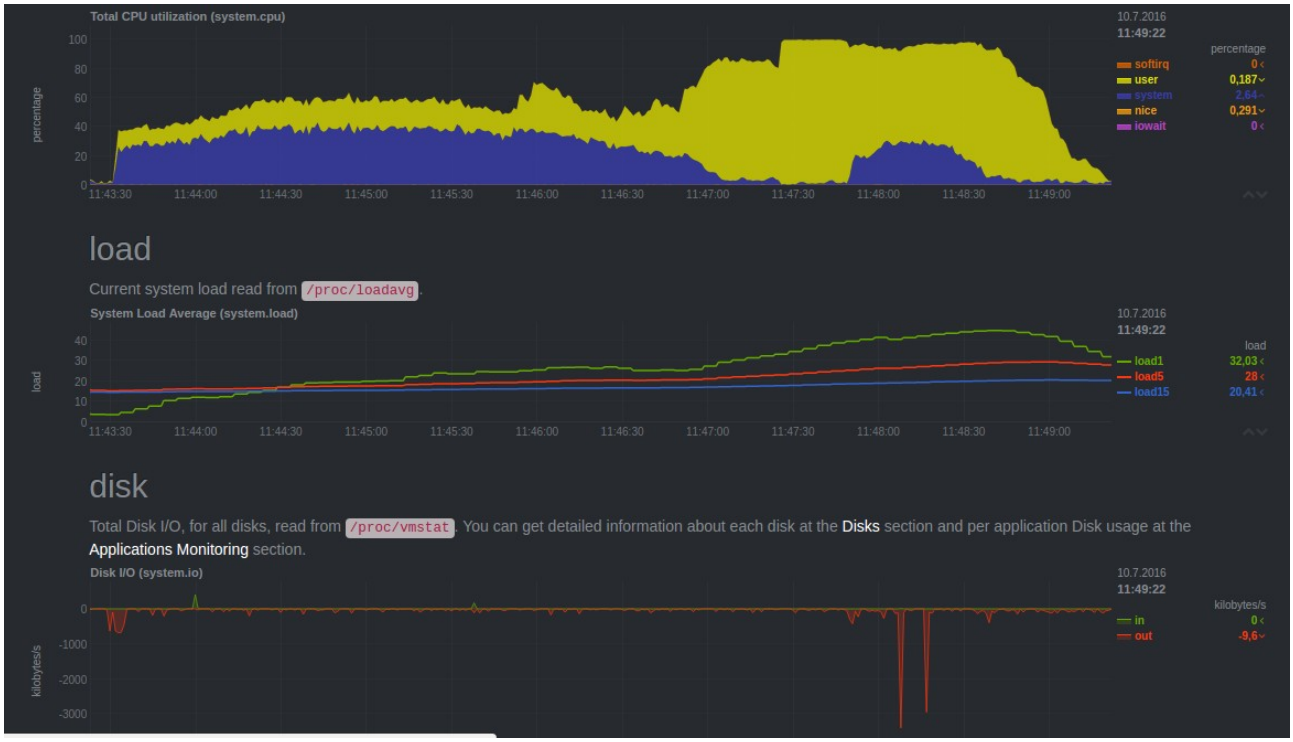
With both the FS and PG caches cold, there are phases with massive storage read IO (> 2.3GB/s). Here, data is read from the NVMe's and brought into the Linux FS and further mapped into the PG shared buffers. During these phases, the system CPU load is also significant. This is expected, as storage IO will run kernel (FS and driver) code.

Here is a snapshot of Linux perf top monitoring a PG background process of a SQL balancer worker running a work unit of the test job:

High system load:

High storage IO:


PG-Cold / FS-Hot

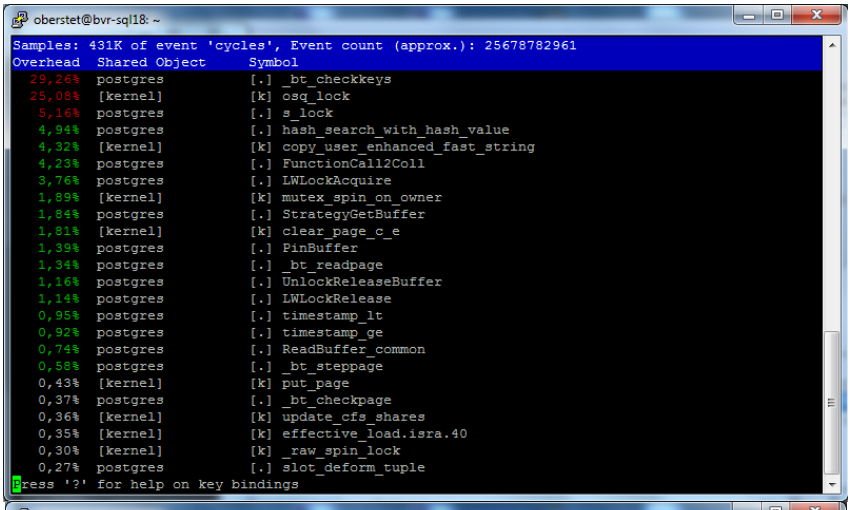
Here is the CPU/disk load during a complete run of the test job with PostgreSQL cache cold and Filesystem cache hot:



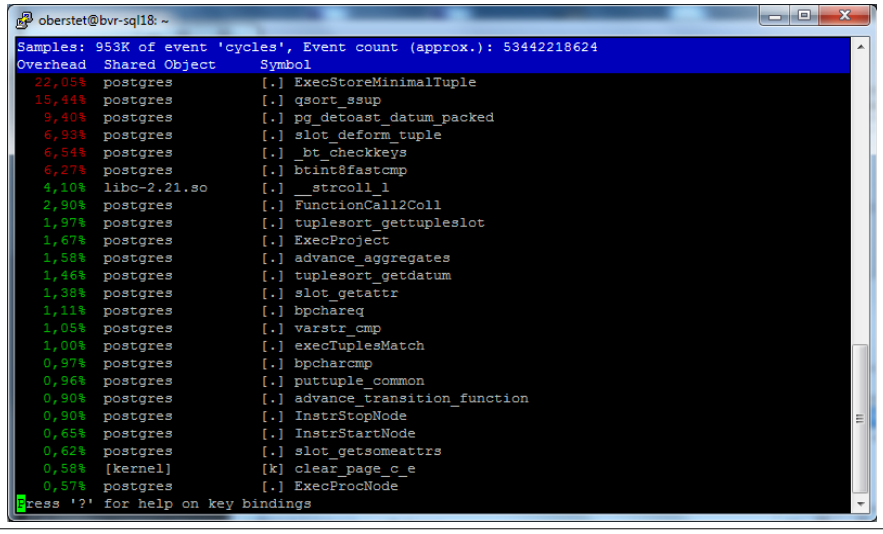
The CPU load total and breakdown between user/system CPU look nearly the same as with cold FS caches. However, there is no IO at all. This makes sense as data is fully cached in FS caches already. And it demonstrates that PG will first need to bring data into PG caches for processing.

Here is a snapshot of Linux perf top monitoring a PG background process of a SQL balancer worker running a work unit of the test job:

Phase 1: bringing data in PG caches
oberstet@bvr-sql18: ~ Samples: 18K of event 'cycles', Event count (approx.): 11299903376 Overhead Shared Object Symbol 54,92% [kernel] [k] osq_lock 6,56% postgres [.] s_lock 4,30% [kernel] [k] copy_user_enhanced_fast_string 2,56% postgres [.] slot_deform_tuple 2,38% [kernel] [k] clear_page_c_e 1,73% postgres [.] StrategyGetBuffer 1,73% [kernel] [k] mutex_spin_on_owner 1,47% postgres [.] hash_search_with_hash_value 1,25% postgres [.] ExecMakeFunctionResultNoSets 1,06% postgres [.] timesub.isra.1.constprop.7 1,04% postgres [.] InstrStopNode 0,92% postgres [.] LWLockAcquire 0,63% postgres [.] j2date 0,62% postgres [.] ExecProject 0,61% postgres [.] pg_next_dst_boundary 0,60% postgres [.] timestamptz_pl_interval 0,58% postgres [.] slot_getattr 0,57% postgres [.] timestamp2tm 0,56% postgres [.] InstrStartNode 0,55% postgres [.] ReadBuffer_common 0,46% postgres [.] heap_getnext 0,46% postgres [.] ExecProcNode 0,45% postgres [.] pg_localtime 0,44% [kernel] [k] find_get_entry Press '?' for help on key bindings
Phase 2: crunching data

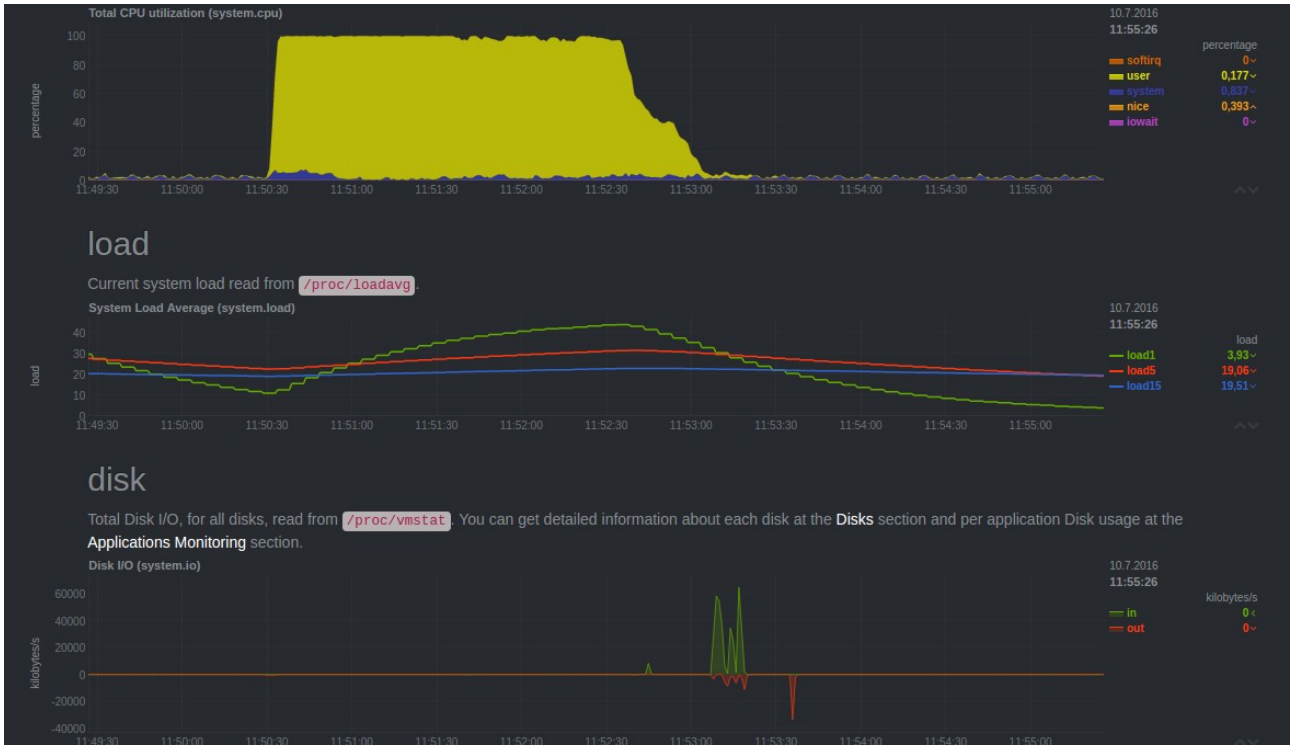


Phase 3: storing results



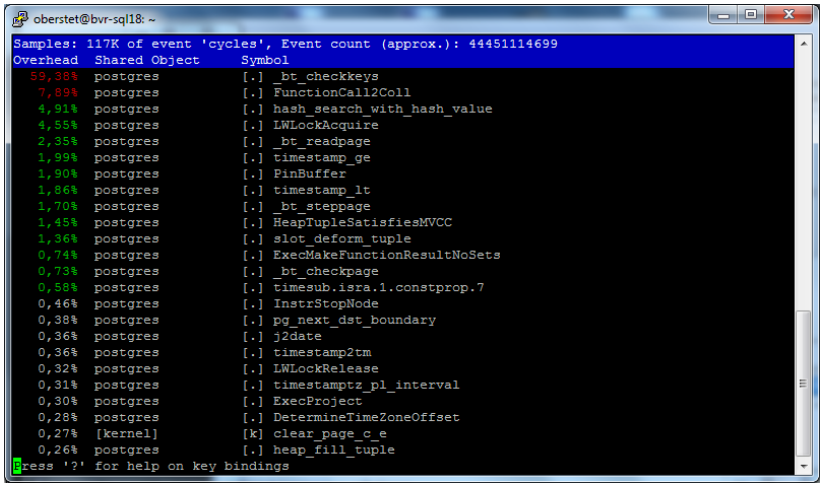
PG-Hot / FS-Hot

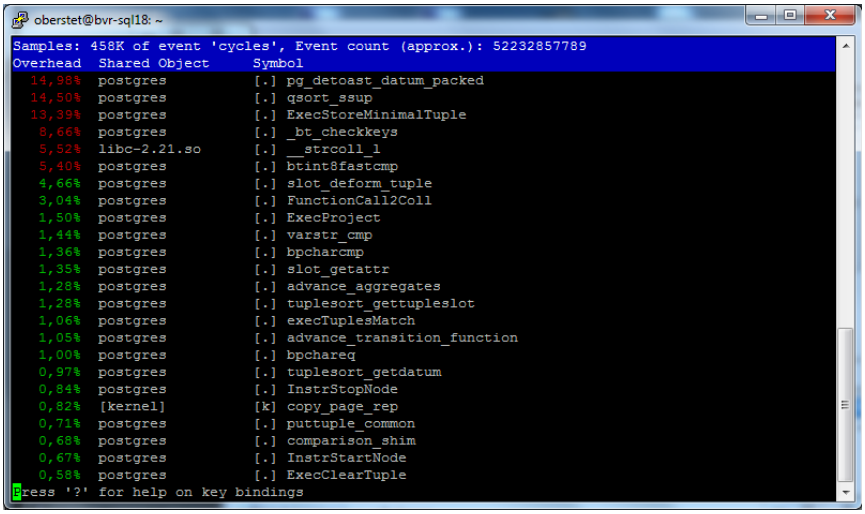
Here is the CPU/disk load during a complete run of the test job with PostgreSQL cache hot and Filesystem cache hot:



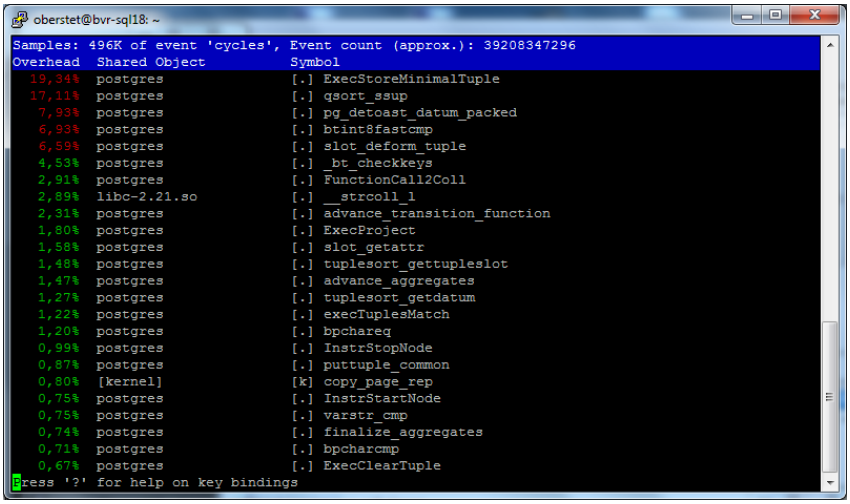
With both the FS and PG caches hot, time is spent almost exclusively in PG code. The specific functions within PG consuming the most time is changing of the run-time of the test job and depend on the phase in which the job.

Here is a snapshot of Linux perf top monitoring a PG background process of a SQL balancer worker running a work unit of the test job:

Phase 1: accessing source data

Phase 2: computing result set



Phase 3: storing results



Conclusion: With data fully cached in PG caches, the whole CPU availably on the machine being spent almost exclusively within PG code, not kernel (or other) code means, that any tuning effort can focus tuning at the SQL level, rather than system (Linux) or storage (FS, hardware).