Scheduler Domains

Each CPU has a "base" scheduling domain (struct sched_domain). The domain hierarchy is built from these base domains via the ->parent pointer. ->parent MUST be NULL terminated, and domain structures should be per-CPU as they are locklessly updated.

Each scheduling domain spans a number of CPUs (stored in the ->span field). A domain's span MUST be a superset of it child's span (this restriction could be relaxed if the need arises), and a base domain for CPU i MUST span at least i. The top domain for each CPU will generally span all CPUs in the system although strictly it doesn't have to, but this could lead to a case where some CPUs will never be given tasks to run unless the CPUs allowed mask is explicitly set. A sched domain's span means "balance process load among these CPUs".

Each scheduling domain must have one or more CPU groups (struct sched_group) which are organised as a circular one way linked list from the ->groups pointer. The union of cpumasks of these groups MUST be the same as the domain's span. The group pointed to by the ->groups pointer MUST contain the CPU to which the domain belongs. Groups may be shared among CPUs as they contain read only data after they have been set up. The intersection of cpumasks from any two of these groups may be non empty. If this is the case the SD_OVERLAP flag is set on the corresponding scheduling domain and its groups may not be shared between CPUs.

Balancing within a sched domain occurs between groups. That is, each group is treated as one entity. The load of a group is defined as the sum of the load of each of its member CPUs, and only when the load of a group becomes out of balance are tasks moved between groups.

In kernel/sched/core.c, trigger_load_balance() is run periodically on each CPU through scheduler_tick(). It raises a softirq after the next regularly scheduled rebalancing event for the current runqueue has arrived. The actual load balancing workhorse, run_rebalance_domains()->rebalance_domains(), is then run in softirq context (SCHED_SOFTIRQ).

The latter function takes two arguments: the runqueue of current CPU and whether the CPU was idle at the time the scheduler_tick() happened and iterates over all sched domains our CPU is on, starting from its base domain and going up the ->parent chain. While doing that, it checks to see if the current domain has exhausted its rebalance interval. If so, it runs load_balance() on that domain. It then checks the parent sched_domain (if it exists), and the parent of the parent and so forth.

Initially, load_balance() finds the busiest group in the current sched domain. If it succeeds, it looks for the busiest runqueue of all the CPUs' runqueues in that group. If it manages to find such a runqueue, it locks both our initial CPU's runqueue and the newly found busiest one and starts moving tasks from it to our runqueue. The exact number of tasks amounts to an imbalance previously computed while iterating over this sched domain's groups.

Implementing sched domains

The "base" domain will "span" the first level of the hierarchy. In the case of SMT, you'll span all siblings of the physical CPU, with each group being a single virtual CPU.

In SMP, the parent of the base domain will span all physical CPUs in the node. Each group being a single physical CPU. Then with NUMA, the parent of the SMP domain will span the entire machine, with each group having the cpumask of a node. Or, you could do multi-level NUMA or Opteron, for example, might have just one domain covering its one NUMA level.

The implementor should read comments in include/linux/sched/sd_flags.h: SD_* to get an idea of the specifics and what to tune for the SD flags of a sched_domain.

Architectures may override the generic domain builder and the default SD flags for a given topology level by creating a sched_domain_topology_level array and calling set_sched_topology() with this array as the parameter.

The sched-domains debugging infrastructure can be enabled by enabling CONFIG_SCHED_DEBUG and adding 'sched_verbose' to your cmdline. If you forgot to tweak your cmdline, you can also flip the /sys/kernel/debug/sched/verbose knob. This enables an error checking parse of the sched domains which should catch most possible errors (described above). It also prints out the domain structure in a visual format.