Memory Saving Techniques – Summary I

Reduction of the memory consumption per event in **current frameworks and tools** in order to **increase throughput.**

Limits: we can probably save of the order of up to 50%

Explored techniques:

- (Late) forking exploiting page-wise copy-on-write
- Worker threads spawned after initialization (requires thread-safeness)
- Kernel SamePage Merging
- Kernel-compressed memory using a virtual swap device
- X32-ABI
- Clever job submission can increase the throughput of a machine

It helps us to **better understand** the application's resource utilization patterns

- Separate read-only allocations from read-write allocations (e.g. caches)
- Investigate why some of the large memory allocations are 0-initialized
- In the course of submitting mixed workloads, we learn about the scalability of applications
- From the swap statistics, we can learn about the applications' memory working set

Memory Saving Techniques – Summary II

Next steps:

- We need to be able to relate the reduction in memory consumption to an increase in throughput

  A common many-core box used to run benchmarks?
- We can explore the possibility of a “HEP-friendly” Linux kernel
  - Supports memory saving features
  - Virtualization-friendly
- Investigate the effort required for the X-32 platform
- Igprof can possibly be extended in order to detect the source of large, zero-initialized blocks