Practice of Software Development: “Dynamic scheduler for scientific simulations”

@ SimLab EA Teilchen
Scheduler is

- a user-friendly middleware between application and cluster scheduling system with graphical (in- and) output interfaces;

- applicable for scientific applications with high number of identical independent tasks in one computational job;

- can be integrated into particular scientific applications:
  - CORSIKA (COsmic Ray SImulations for KAscade) code for simulation of extensive air showers initiated by high energy cosmic ray particles;
  - Simulation code for Atomic Clusters;
  - PolGrawAllSky code searching for periodic gravitational wave signals.
Scheduler modules

Possible system events:

- **executor interface:**
  - send: execute task;
  - receive: task finished.

- **code interface:**
  - receive: new task.

Coupling types:

- task data;
- schedule information;
- communication.

where “task” is a pair of:

- Data Type (e.g. structure, array);
- Input Data of the specified type.
Scheduler modules: advanced version

- GUI – Graphical user interface for the input data entering and batch script generation for particular batch system;
- can be based on X-Win API and batch system API e.g. MOAB API;
**System events**

Possible system events:

### “new task” signal
1. receive new task from code interface;
2. send task to data mining module;
3. analyse task according to collected statistics;
4. send estimated task weight to scheduler;
5. schedule task (place it in the queue);

### “task finished” signal
1. receive finished task from executer interface;
2. store task information to the database;
3. take new task from the queue;
4. send “execute task” signal to executer interface;
Module – scheduling + communicator

Scheduling Module:
- contains a variety of scheduling algorithms and strategies;
- queries Data Mining module, how much resources the task needs;
- makes a decision how to place a task in a schedule (queue / run immediately);
- does bookkeeping;

Communicator:
- has to organize a multi-level communication among multiple parallel schedulers (one computation – several processes – multiple tasks);
- communication can be organized e.g. as following:
  - submit new multiprocessor jobs;
  - change the size of your parallel world;
  - fix the size of parallel world and distribute tasks among reserved processes;
Ordering resources when required

- submit new multiprocessor jobs using shell script generation or cluster submission system API (e.g. Moab-API):
  - batching jobs @bwHPC clusters;
  - Interfacing with Moab (APIs);

- use MPI 3.x mechanism to change the size of you parallel world (the number of processes you are using for computation) according to current requirements:
  - MPI 3.1 Standard;
  - MPI-dynamicprocesses.pdf;
  - “advanced topics in MPI programming” @etutorials, chapter 9;
Fixed amount of resources (size of MPI World)

- Single-queue master-managed scheduling:
  - one process is a “master” and does the scheduling,
  - others are “slaves” and do the work;

  - “master” is a bottleneck and can serve only one slave at a time;  
  + relatively easy to implement;

- Multiple-queues scheduling (preferable)
  - each process has its own queue of tasks,
  - tasks can be re-distributed among queues;

  - difficult to implement – synchronisation “many-to-many” should be organized;  
  + no bottlenecks – all processes are doing equal algorithm;
Single-queue master-managed scheduling

Each process can request master to access the queue

Rules can be:
- LIFO, FIFO;
- Long-first, Short-first;
- Round-Robin;
- Statistics-based.

See more about scheduling strategies in the following Doctoral theses:
- chapter 2.2 from shed2.pdf (good basic overview);
- chapters 2 and 3 from shed1.pdf;
- chapters 2.1 - 2.4 from shed3.pdf;
Multiple-queues scheduling

Each process has its own queue, and does the same task:

- Task queues
- Processes

The queue managing can be one of the following:

- Random or rule-based task distribution;
- Statistics-based task distribution;
- Task stealing (see next 2 slides).
Task stealing algorithm (equal for all processes)

begin

check own q

no task

for ∀ p in other processes

all procs are idle

no

no task

check p's q

task

work (task)

end

Blocking operation on own queue. Check whether there is a task and if yes - take it (and remove form q).

Blocking operation on p’s queue. Check whether there is a task and if yes - take it (and remove form q).

Work cycle: the “execute” function of the executer interface with input parameters taken from task is called here. During the work cycle one or several new tasks may be added to the own queue.
**Task stealing algorithm – intermediate state**

Some processes are working, some are stealing;

How to organize “stealing” on distributed-memory systems see in:
- Cornell Virtual Workshop on MPI one-sided communication;
- The article “One-Sided Communications with MPI-2” in Linux Magazine;
Module – Data mining

- is called from Scheduling Module;
- analyses the statistics to define resource requirements for a task;
- keeps statistics for completed tasks;
- accumulates statistics of the equal computations (same scientific code) in a single database.

Advanced:
- polynomial approximation of collected statistics (e.g. once at the end of computation);
- using single function instead of on-line access to the database and data analysis on each step;
Module – Database

- is used to keep statistics, bookkeeping and local schedule;
- can either be a parallel-accessed or multiplied among processes with synchronisation during the computational process or accumulation at the end of it;

- Bookkeeping is done for the following events:
  - task appears (timestamp, task parameters, parent process);
  - task started (timestamp, task parameters, hosting process);
  - task finished (timestamp, task parameters, hosting process);
  - interconnection between processes (timestamp, processes’ ranks, result);

- Statistics collected for each task is:
  - the runtime of the task depending on task parameters.
Module – Visualization system

Visualization system is used to visually analyse the collected data.

Either

- exports obtained results into selected format (e.g. Gnuplot format, .vtk format for Paraview or any other preferred format, used by open-source visualization software);

or

- works with visualization system API (e.g. Gnuplot API).
Code and executer interfaces

The “task” is a set of two:

- **Data_Type** (e.g. structure or array);
- the **Data** of this type;

Within one computation the Data_Type is fixed, differs only the Data. However, interfaces to different simulation software have different task types. Therefore template programming usage is highly recommended.

Simple example can be found at: