Advances in VT’s Load Balancing Infrastructure and Algorithms

Team (alphabetically):

Jakub Domagala (NGA)
Ulrich Hetmaniuk (NGA)
Jonathan Lifflander (SNL)
Braden Mailloux (NGA)
Phil B. Miller (IC)
Nicolas Morales (SNL)

Ceaward Skrzynski (NGA)
Nicole Slattengren (SNL)
Paul Stickney (NGA)
Jakub Strzeboński (NGA)
Philippe P. Pébaÿ (NGA)

NGA = NexGen Analytics, Inc
SNL = Sandia National Labs
IC = Intense Computing
### What is DARMA?

A toolkit of libraries to support incremental AMT adoption in production scientific applications

<table>
<thead>
<tr>
<th>Module</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DARMA/vt</td>
<td>Virtual Transport</td>
<td>MPI-oriented AMT HPC runtime</td>
</tr>
<tr>
<td>DARMA/checkpoint</td>
<td>Checkpoint</td>
<td>Serialization &amp; checkpointing library</td>
</tr>
<tr>
<td>DARMA/detector</td>
<td>C++ trait detection</td>
<td>Optional C++14 trait detection library</td>
</tr>
<tr>
<td>DARMA/LBAF</td>
<td>Load Balancing Analysis</td>
<td>Python framework for simulating LBs and experimenting with load balancing strategies</td>
</tr>
<tr>
<td>DARMA/checkpoint-analyzer</td>
<td>Serialization Sanitizer</td>
<td>Clang AST frontend pass that generates serialization sanitization at runtime</td>
</tr>
</tbody>
</table>

Load Balancing R&D Lifecycle

- Application runs with VT runtime with designated *phases* and *subphases*
- VT exports LB statistics files containing object loads, communication, and mapping
- LBAF loads the statistics files, and simulates possible strategies
  - LBAF analyzes the mapping and can produce a new mapping with an experimental LB implemented in Python
  - LBAF exports a new set of mapping files
- The application can be re-run with *StatsMapLB* to follow the LBAF-generated mapping and measure the actual impact
- Process can be iterated, shortening LB development and tuning cycle
Phase Management

- A phase is a collective interval of time over all ranks that is typically synchronized
  - In an application, a phase may be a timestep
  - In VT parlance, a phase will often be a “collective epoch” under termination detection
  - Load balancing in VT fundamentally operates over phases

- A phase can be broken down into subphases
  - A subphase is typically a substructure within a phase of an application’s work that has further synchronization
  - Creates vector representation of workload

- We have explored the idea of further ontological structuring for the purpose enriching LB knowledge, but so far have only implemented phases and subphases
Phase Management

- Building general interface for general phase management
- Many components can naturally do things at phase boundaries
  - LB
    - Running a strategy (or several) and migrating objects accordingly
    - Outputting statistic files
  - Tracing
    - Specifying which phases traces should be enabled for which ranks
    - Specifying phase intervals for flushing traces to disk
  - Memory levels/high-water watermark for runtime/application usage
  - Diagnostics
    - Just finished developing a general diagnostic framework for performance counters/gauges of runtime behavior (e.g., messages sent/node, bytes sent/node, avg/max/min handler duration)
- Checkpointing of system/application state
- Termination
  - Recording state of epochs for debugging purposes
Phase Management

- A *phase* is a collective interval of time over all ranks that is typically synchronized
  - In an application, a phase may be a timestep
  - In VT parlance, a phase will often be a “collective epoch” under termination detection
  - Load balancing in VT fundamentally operates over phases

- A phase can be broken down into *subphases*
  - A subphase is typically a substructure within a phase of an application’s work that has further synchronization
  - Creates *vector* representation of workload

- We have explored the idea of further ontological structuring for the purpose enriching LB knowledge, but so far have only implemented phases and subphases
EMPIRE Load Structure – Phases, Subphases, Iterations
Subphase Vector Loads

\[
t = \sum_s t_s \quad t_s = \max_p w_{ps} \quad W = AL \quad L: \mathbb{R}^{N \times S} \quad A: \mathbb{B}^{P \times N}
\]

Total Time \quad \text{Subphase Times} \quad P \times S \quad \text{Object Loads} \quad \text{Object Assignments}

\[\forall n \left[ \sum_p a_{pn} = 1 \right]\]

Objective Function:

\[
\min_A t
\]
Subphase Vector Loads

- From 0-1 optimization to smaller Integer Program optimization
  \[ a_{pn} = 1 \iff m_n = p \]

- Replace with to (partially) linearize

- Plug this in to standard solvers
  - Possibly MPI-based for live use!

\[ t_s = \max_p w_{ps} \quad \text{with} \quad \forall_p [t_s \geq w_{ps}] \]
Load Modeling

- When a selected strategy runs after a phase completes, it has access to data from the application’s execution
- *Load models* provide a novel mechanism for manipulating how the load balancer observes instrumented data from phases and subphases, past and future
  - The most basic, naïve model would read raw instrumented data and assume it persists to the next phase/subphase to perform task assignment calculations for the subsequent phase
  - Explicit embodiment of “principle of persistence”
  - Offers configuration, alternatives
  - Composable functions, easy extension

- Can also map vector of per-subphase data to scalars for current strategies
struct PhaseOffset {
    int phases;
    static constexpr unsigned int NEXT_PHASE = 0;
    unsigned int subphase;
    static constexpr unsigned int WHOLE_PHASE = ~0u;
};

class LoadModel {
    virtual TimeType getWork(
        ElementIDType object,
        PhaseOffset when
    ) = 0;
    // ...
};

Default:
NaivePersistence . Norm(1) . RawData
## Load Balancing Strategies

<table>
<thead>
<tr>
<th>Load Balancer</th>
<th>Type</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>RotateLB</td>
<td>Testing</td>
<td>Rotate objects in a ring</td>
<td>vt::vrt::collection::lb::RotateLB</td>
</tr>
<tr>
<td>RandomLB</td>
<td>Testing</td>
<td>Randomly migrate object with seed</td>
<td>vt::vrt::collection::lb::RandomLB</td>
</tr>
<tr>
<td>GreedyLB</td>
<td>Centralized</td>
<td>Gather to central node apply min/max heap</td>
<td>vt::vrt::collection::lb::GreedyLB</td>
</tr>
<tr>
<td>GossipLB</td>
<td>Distributed</td>
<td>Gossip-based protocol for fully distributed LB</td>
<td>vt::vrt::collection::lb::GossipLB</td>
</tr>
<tr>
<td>HierarchicalLB</td>
<td>Hierarchical</td>
<td>Build tree to move objects nodes</td>
<td>vt::vrt::collection::lb::HierarchicalLB</td>
</tr>
<tr>
<td>ZoltanLB</td>
<td>Hyper-graph</td>
<td>Run Zoltan in hyper-graph mode to LB</td>
<td>vt::vrt::collection::lb::ZoltanLB</td>
</tr>
<tr>
<td>StatsMapLB</td>
<td>User-specified</td>
<td>Read file to determine mapping</td>
<td>vt::vrt::collection::lb::StatsMapLB</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- Increase expressiveness of load data
- Shorten LB development and tuning cycles
- Improve abstractions in real implementations
- Formalize time-vector balancing challenge
  - Can actually try out dedicated solvers and general heuristics