Optimization of the HLT Resource Consumption in the LHCb Experiment

M.Frank, C. Gaspar, E v. Herwijnen, B. Jost, N. Neufeld, R. Schwemmer
CERN, 1211 Geneva 23, Switzerland
E-mail: Markus.Frank@cern.ch

Abstract. Today's computing elements for software based high level trigger processing (HLT) are based on nodes with multiple cores. Using process based parallelization to filter particle collisions from the LHCb experiment on such nodes leads to expensive consumption of memory and hence significant cost increase. In the following an approach is presented to both minimize the resource consumption of the filter applications and to reduce the startup time. Described is the duplication of threads and the handling of files open in read-write mode when duplicating filter processes and the possibility to bootstrap the event filter applications directly from preconfigured checkpoint files. This led to a reduced memory consumption of roughly 60 % in the nodes of the LHCb HLT farm and an improved startup time of a factor 10.

1. Introduction

LHCb is a dedicated B-physics experiment at the LHC collider at CERN [1]. LHC delivers proton-proton collisions at a centre of mass energy of up to 14 TeV to the LHCb detector at a rate of 40 MHz. LHCb is designed to exploit the finite lifetime and large mass of charmed and beauty hadrons to distinguish heavy flavor particles from the background in inelastic pp scattering. The first level trigger, which reduces the rate of accepted events to 1 MHz, is hardware based and located in the frontend electronics. The second level or High Level Trigger (HLT) is purely software based. As shown in figure 1, the front end readout boards (TELL1 boards) send data from particle collisions at a rate of roughly 1 MHz through a switching network to the HLT farm nodes, where dedicated algorithms compute the decision on whether the event is to be accepted. Events with a positive decision are sent to the storage system and subsequently to the GRID for later offline analysis. The HLT hardware consists of roughly 1500 dual processor units grouped to 56 subfarms (see figure 1), which host about 29000 trigger processes.

The experiment controls system (ECS), implemented using the commercial product PVSS [2], handles the configuration, monitoring and operation of all experimental equipment. All macroscopic entities of the ECS are modeled as Finite State Machine (FSM) entities, grouped together in a tree structure. The state of every higher level node summarizes the state of its children. The same FSM tree mechanism is used to describe the functioning of the processor farms such as for example the HLT processor farm, where at the lowest level processes are modeled as FSM elements, a set of processes is grouped to a node, a set of nodes describes a subfarm and finally as the set of subfarms represents the high level trigger.

The HLT processes are configured simultaneously at the beginning of the data taking activity. The transitions between the states of the FSM are used to configure the HLT processes according to the internal state diagram shown in figure 2. Figure 3 shows the different types of processes executing on a HLT worker node:

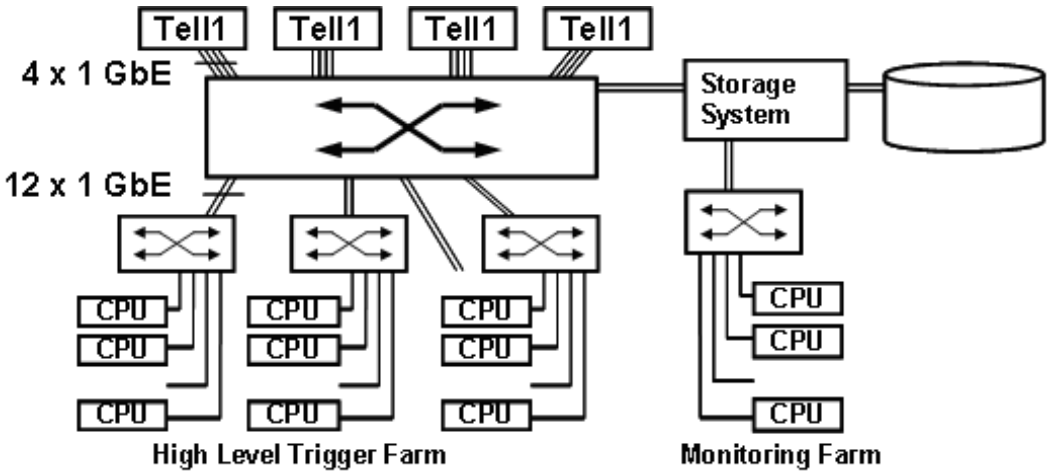

Figure 1. The layout of the LHCb DAQ hardware. The data from particle collisions are sent from the fronend boards (Tell1) through a switching network to the worker nodes of the HLT processor farm (CPU).

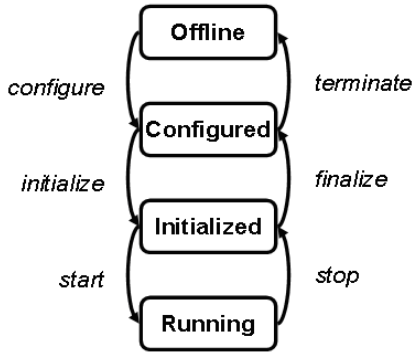

Figure 2. The state diagram of a Gaudi process. Every process starts in the Offline state. Three transitions are performed to reach the running state, where the process is fully configured and is able to process events from particle collisions in the LHC. To shutdown a process a reverse triplett of transitions is necessary, where each transition cancels the actions performed in the corresponding transition to the Running state.

- The *EventBuilder* reads network packets sent from the frontend boards and assembles event data to contiguous blocks, which are declared to the *Events* buffer.
- The actual trigger processes *Moore* analyze the physics data in these events and declare events to be stored for later offline data analysis to the *Send* buffer.
- The *Sender* registered to the *Send* buffer sends the accepted events to the long term data storage.

Out of these processes only the *Moore* process is complex, it containes all physics knowledge necessary to identify rare quark decays LHCb is designed to identify. The other processes do not consume many resources and start quickly. The HLT trigger processes are rather complicated software constructs. This software trigger consists of many small components, which require individual configuration and data input such as the magnetic field map, the detector geometry and the detector conditions. The

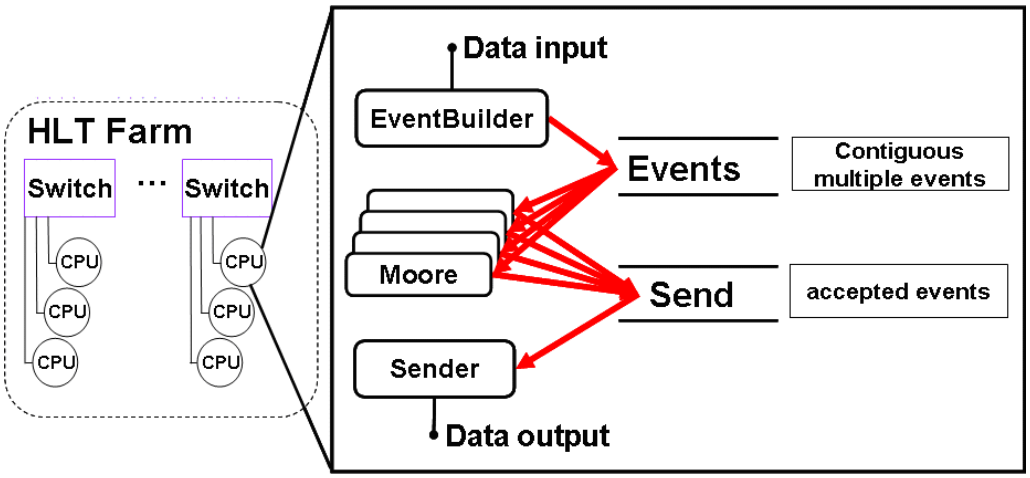

Figure 3. The event data flow between the various processes executing on the worker node. Resource consuming are only the instances of the HLT trigger process Moore.

configuration of such a process is a time consuming procedure, which accesses numerous resources. Also the memory footprint created when interpreting this information is significant. The mechanism to reduce this configuration time and to optimize the resource usage is presented in the following sections.

2. Problem Analysis

The time consumption to configure HLT processes and the corresponding data input led to startup times of the data taking activity for the entire experiment of more than 20 minutes, where a large fraction was used to start the HLT. Such operations are clearly much too long to quickly respond to situations, which require a HLT restart during data taking, since such a restart implied a significant loss of valuable beam time and thus a loss of physics data. Investigations on the running system led to the conclusion that a possible solution to this problem must address the following three issues:

(i) Minimize the amount of network based (NFS) file accesses.

(ii) Minimize the integrated amount of memory used by the trigger processes.

(iii) Minimize the time a single trigger process requires until it is able to process events.

The HLT filter code is based on the Gaudi data processing framework [3], which provides the basic functionality to the different software components of an HLT process. Any solution must preserve the functionality provided by the Gaudi framework. The following sections address solutions to minimize the resource usage mentioned above.

3. Solution Development

The nature of the scarce resources: disk access, memory and startup time is quite different. The different nature also suggests that the different solutions to be developed in order to minimize the resource usage are rather unrelated. However, possible interferences of the different approaches, which are discussed here have to be closely monitored. Possible solutions may not be destructive, but rather help to boost individual benefits. Another important consideration to be taken into account is the fact that the HLT software already exists. Any improvement must be as transparent as possible to existing code. Significant changes to the existing code base are not possible. In the next three section these solutions are discussed in detail.

3.1. Minimizing Network File Access

Each HLT trigger process accesses during the configuration step several hundred MB of configuration data and maps several hundred MB of library images. The individual components are configured with python [4], which loads many small precompiled files. Hence, a mix of many small and several large files has to be read by every process. However, not only the actual file access, but also the file lookup from the nfs server is not at all negligible. Here the limiting factor is the latency introduced by the network communication to a highly loaded nfs server.

Most of the files accessed are static, they do not change between consecutive restarts, such as the magnetic field map, the conditions in a SQLite database file and the software libraries. Only a small fraction of the library code is subject to change. These are mostly urgent updates and bug-fixes. Read-only files in each node are accessed from a read-only nfs-mount point. Since all trigger processes start simultaneously, they access the files at the same time. This leads to a high load of the nfs servers and thus to a decreased performance seen by worker nodes due to limitations of the network bandwidth. Some requests even have to be served multiple times due to internal timeouts and retry attempts.

An obvious solution is to cache these files. The cache mechanism is based on a fuse file-system [5], which starts up empty. On the first access the file is copied from the network location to a local ram disk. Since the same version of an HLT process is normally restarted many times, subsequent file accesses or file lookups avoid any network transfer and response is nearly immediate. Such a local lookup on the ram disk introduces no latency. This mechanism significantly eases the load of the nfs servers and hence improves the configuration time.

3.2. Minimizing the Physical Memory Usage

One single HLT process on a farm node has a memory footprint of roughly 1.3 GB before any event processing activity starts. Once processing events, the resident memory size stabilizes at a maximum of roughly 1.7 GB. For a 12 core machine with hyper-threading [6] enabled, thus a total for 24 processes executing, this would lead to a memory consumption exceeding the available physical memory of 2 GB per core. This memory size was enough at the time the machines were ordered and more physical memory per core would have led to a worse cost effectiveness and hence to higher costs of the farm. Since, the trigger processes became more complex and increased in size. However, roughly 60 % of the memory used by each HLT trigger process is written once and accessed in read-only mode afterwards. Memory objects representing the detector geometry, the description of the magnetic field etc. are never modified during the lifecycle of a process. The Linux operating system allows to share read-only memory section between processes, if these have a common ancestor. Thus, after forking, child processes allocate physical memory pages from the operating systems memory pool only if a page is modified. Hence, child processes should only be forked once the parent process is fully initialized. The parents initialization phase can take up to several minutes. The child processes then subscribe to the buffer manager [7] to receive events and apply the trigger code. The mechanism to fork identical child processes executing the HLT code must preserve the following process properties:

(i) All existing process threads must be preserved, in particular the internal state of the thread libraries. Threads are not replicated when the child processes are forked.

(ii) Open file handles must be preserved. By default after a fork the parent's and the child's file handles are shared, which is undesired. Temporary files, which are already unlinked must be replicated for each child.

(iii) The initial parent process and, after the fork the child processes are controlled by the ECS [8]. The ECS related communication layer [9] must be re-established.

The existing open source software package MTCP (Multithreaded Checkpointing) [10] solves the first two problems. It could be re-used with only minor modifications. As described in [10], to halt all threads requires the following actions:

(i) To ensure all child processes are properly controlled by the ECS after creation, the communication layer is closed

(ii) A signal is sent to all child threads using the Linux system call tkill [10].

(iii) The state of each thread (registers, thread local storage) [10] must be saved.

(iv) All child threads are suspended from execution by waiting on a Futex cell [10].

(v) Save all open file descriptors, the corresponding state and the buffers of temporary files.

Forked children inherit the process environment from their parent. In order to install a process specific environment like a process name, which is used to deliver event data to the process or to steer the process state from the ECS, the environment must be explicitly set before each fork and restored after all children are created. The child processes must be forked using the Linux fork system call - the C runtime library call may not be used, it would reset the state of the POSIX thread library, which applies fork handlers. After the fork, each child restores the file access, recreates the threads and re-establishes the communication layer. From this point onward the child is steered by the ECS. The parent process acts as a watch-dog to its children and in case of a premature end it forks a replacement of a crashed child using the same procedure.

A positive side effect arises from the fact, that forked children do not have to access directly the data and the libraries used during configuration. This reduces the overall disk accesses by an average factor 20.

3.3. Minimizing the Initialization Time

Ideally, the trigger processes would not require any configuration and hence the initialization phase would be very fast. Directly, after invocation, all constants should be present in the filter code. Such an approach is possible, but is highly inflexible i.e. the change of any parameter constant inevitably would requires a rebuild of the executable. Such an approach clearly is neither practical nor maintainable in the presence of a large software base.

Instead of all programming constants being compiled into the executable image, an alternative valuable solution is to completely initialize one HLT process and create a dump of the entire process to a checkpoint file. Any subsequent process restart would read the checkpoint file and the process starts execution from the state the process was saved. This solution was adopted for the LHCb HLT. The check-pointing mechanism is a logical continuation of the developments necessary to implement the process forking mechanism (see Section 3.2); the five steps necessary before a child can be forked from the parent must as well be performed before a consistent checkpoint can be written. For completeness the basic principles described in [10] are repeated here. The checkpoint file contains:

- The complete environment of the parent process to be restored.
- The information to restore the state of all open file descriptors. For temporary files even if already unlinked - the actual file content must be saved.
- The information to restore all mapped memory sections including the Linux heap and stack segments. If the memory sections are readable, the data content must be saved.
- The binary code necessary to restore the file descriptors and the memory mappings.
- For restore optimization all linked shared libraries can optionally be cached in the checkpoint file. Hence, the checkpoint is complete and does not require further images unless a late dynamic load would occur after the restore sequence. This is not the case for the LHCb HLT.

The process restart is initiated by a statically linked executable. Static linking is necessary to avoid address clashes with libraries such as the C runtime library. The restoration of the original process then requires the following actions [10]:

- The checkpoint file is opened and all contained shared libraries are restored to a temporary directory.

- The environment of the original process is restored within the local process.
- The restore process re-executes its image to ensure the layout of the original stack frame at the end of the address space.
- The restore process maps the binary code required for the restoration at exactly the same address range present at checkpoint time. All other memory areas are unmapped.
- The process executes the mapped restore code, switches to an internal stack area contained in the address range of the restore code.
- The rest of the memory contents from the checkpoint file is read from the file and all sections are mapped to exactly the same address ranges they occupied at the time the checkpoint was written.
- Once restored, the process switches back to the original stack.
- Finally, if needed, child processes are forked and file descriptors and threads are restored.

This mechanism of creating a checkpoint file and restarting HLT processes from such a checkpoint file is handy and works reliable. Checkpoints are created using a small graphical application that allows to set all parameters, which make the checkpoint unique. A snapshot is shown in figure 4. The size of a typical checkpoint file, which includes the necessary libraries of roughly 1.8 GB is unfortunately quite large and has to be distributed to all 1500 worker nodes of the HLT farm at the start of run. This means a total data volume of roughly 2.7 TB must be distributed within a time frame short compared to the overall configuration of the experiment at the beginning of a data taking activity, i.e. less than one minute. Such data rates are beyond the capabilities of the nfs servers and the storage system. File compression leads to a reduction by a factor 5 and a file size of 350 MB, which is still too large. Though, we can benefit from the fact that the identical file must be replicated, which opens the possibility to use other replication mechanisms rather than a plain file copy. The need to efficiently replicate a large file many times is common to many file-sharing networks. Hence, it is natural to also use such a technology here.

In analogy to the HLT hardware configuration, where the worker nodes are grouped in 56 subfarms steered by a common control, a tree of file loaders was installed, which would download the checkpoint file to the local node. The downloader is based on the *BitTorrent* transfer protocol [11]. The *BitTorrent* protocol enables every client, which either has a complete copy or fragments of the requested file to act as a server for numerous other clients. Given a sufficient number of clients the integrated download rate rises exponentially and becomes network limited. In fact, to not overload the experiment controls network, not all nodes are allowed to communicate with each other.

From the startup script, the HLT trigger process would issue a request to the local loader on the worker node. This request is then forwarded to the loader residing on the sub-farm controls node, which again forward the request to the primary file loader with access to the created file. The primary file loader serves as a seed to all subfarm controls nodes. Each of the subfarm controls nodes is allowed to seed the worker nodes of its own connected subfarm and the worker nodes of two other subfarms. Once downloaded, the subfarm controls nodes cache the download data on the local disk.

4. Practical Experience

The three independent optimization procedures described above are used in the LHCb HLT trigger. The procedures work very well together and in total allowed to have a fully scalable trigger farm with a fast configuration step. Starting with an initial configuration time of roughly 10 minutes, the following improvements were seen:

- The file system optimization using a local cache was the first improvement made. This improvement ensures during normal running, that the configuration time only depends on the CPU needs of the trigger process and does not depend on the number of nodes participating in data taking. The overall configuration time was reduced by a factor bretween 2 and 4.
- The forking mechanism not only saved roughly 60 % of the physical memory, but also allowed to over-commit individual worker nodes, i.e. more jobs are ececuting on each node than physical CPU

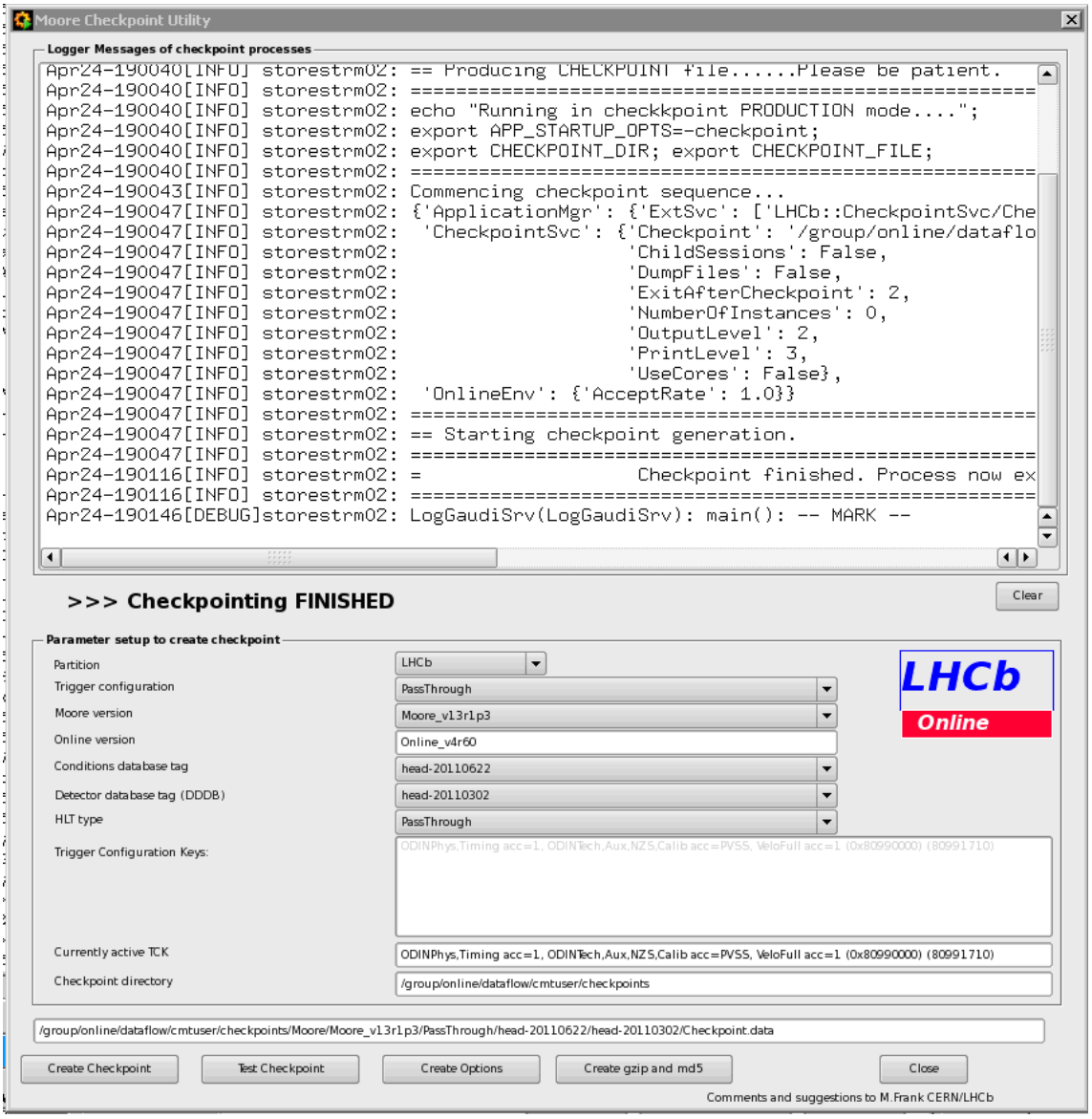

Figure 4. Snapshot of the checkpoint creation tool. The tool allows to generate checkpoints, test the produced checkpoint, to compress the checkpoint file and to create the BitTorrent descriptor.

cores present. In this regime processes would have to share one core during the initialization phase and hence would starve and slow the configuration down even more. Beneficial however is that the overcommitment helps to minimize the average time any CPU core is idle due to a process waiting for I/O operations. Since only one process instance needs to be configured per physical node, also the configuration time is reduced.

- Restarting HLT processes from a checkpoint file requires in average less than two minutes for the entire HLT farm, another improvement by a factor 2.5. The download of a newly created checkpoint file takes O(1 minute). Once present the download time is reduced to O(10 seconds) and thus the overall configuration further reduced.

In total the overall configuration time was reduced by more than a factor of 10, which is well within the budget, defined by the second slowest component to be configured at the beginning of a data taking

activity.

5. Conclusions

A mechanism was designed and implemented to improve the startup time and the memory consumption of the LHCb HLT trigger processes. Roughly 60 % of the memory can be shared between the processes in one node and the initialization time was reduced by more than a factor of 10. This achievement not only minimizes the time necessary to configure the experiment before particle collisions occur in the LHC collider, but also allows to react to data taking problems faster and solve problems quicker in case they require the restart of the trigger processes. It was demonstrated that such improvements can be applied retroactively to an existing software framework.

References

[1] LHCb Collaboration, LHCb the Large Hadron Collider beauty experiment, reoptimised detector design and performance CERN/LHCC 2003-030

[2] PVSS-II, [Online]. Available: http://www.pvss.com

[3] G.Barrand et al, GAUDI : The software architecture and framework for building LHCb data processing applications, Comput. Phys. Commun. 140 (2001) 45-55

[4] V.Gligorov et al, The HLT inclusive B triggers, LHCb-PUB-2011-016

[5] R. Schwemmer et al, Launching large computing applications on a disk-less cluster, 2011, Journal of Physics: Conference Series, Vol. 311 052032

[6] Hyper-Threading Technology, Intel Technology Journal, Vol.06, Issue 01, Feb 14, 2002, ISSN 1535766X

[7] M.Frank et al., The LHCb High Level Trigger Infrastructure, International Conference on Computing in High Energy and Nuclear Physics (CHEP), Victoria, Canada, 2 - 7 Sep 2007

[8] E.van Herwijnen, Control and monitoring of online trigger algorithms using a SCADA system, 15th International Conference on Computing In High Energy and Nuclear Physics (CHEP), Mumbai, India, 13 - 17 Feb 2006

[9] C.Gaspar et al, DIM, a portable, light weight package for information publishing, data transfer and inter-process communication Computer Physics Communications 140 1+2 102-9.

[10] J.Ansel, G.Cooperman, M.Rieker, Transparent User-Level Checkpointing for the Native POSIX Thread Library for Linux, The 2006 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'06), Las Vegas, NV. Jun., 2006.

[11] A. Norberg et al., Rasterbar software: libtorrent, http://www.rasterbar.com/products/libtorrent