**Technical Group Leader Position Proposal Concept**

**due: Sept 25th, 2020**

**Title**: Using transparent checkpointing to facilitate fault tolerance and Navigational Programming

**Proposer**: Lei Pan

**Background**: In the currently available science data systems (SDS), there are two problems that are not properly addressed: (1) There is no fault tolerance mechanism for SDS to leverage Amazon EC2 Spot Instances, which are spare compute capacity in the AWS cloud available at steep discounts (90% savings) but can be taken away at any time. In the meanwhile each atomic task can take as much as 3-5 hours to finish. Fault tolerance would fully leverage computation resources at very low cost; and (2) The distributed parallel system is not directly programmable by scientist programmers. One would always have to work with an SDS expert in virtualizing and deploying the app level programs even between version updates. For applications that are not by nature embarrassingly parallel, this task is much more difficult if not impossible.

**Objective**: We will introduce an open source transparent checkpointing system and enable navigational programming to facilitate high performance, effective resource leveraging, and ease of use for scientist programmers.

**Approach**: We will leverage the DMTCP system (Distributed MultiThreaded CheckPointing) developed by Professor Gene Cooperman of Northeastern University [1]. DMTCP transparently checkpoints both single-host and distributed computation in user space. It works on most Linux applications, including python/numpy, C/C++/Fortran, Matlab, Perl, R, MPI, OpenMP, and many programming languages. DMTCP requires no modifications to user application code nor to the OS kernel, and it has very low overhead of well less than 1% in most cases. It is robust and widely used at Intel, Microsoft, and the Argone national lab, to name a few.

In a nutshell, DMTCP saves a copy of the state of a running process to disk (called a checkpoint memory image) and resumes the process later wherever the memory image goes. So in the case when an EC2 Spot Instance unplugs in the middle of computation, the checkpoint memory image can be brought to another EC2 instance to resume computation, thereby saving hours of CPU time. To do this, three dmtcp commands are involved: (1) dmtcp\_launch ./myapp; (2) dmtcp\_command --checkpoint (run everything 10 minutes for example); and (3) dmtcp\_restart <ckpt mem image>. So in this example, we lose at most 10 minutes of wall-clock time if the EC2 Spot Instance unplugs on us.

DMTCP facilitates process migration and process replication for applications in many programming languages. This can be used to enable Navigational Programming (NavP) [2,3]. NavP is a programming paradigm for distributed parallel computing. It is based on the principle of “self-migration.” A programmer controls the locus of computation by inserting hop(dest) statements in the program [2]. The execution of a hop(dest) statement pauses the computation, encapsulates the program state, moves to the dest node, restores the state, and resumes the computation. The hop(dest) statements provide details of where the locus of computation will be, but in terms of code structure, the code looks exactly the same as the original sequential algorithm, making program development and maintenance straightforward. Multiple NavP computations can be used to build “mobile pipelines” to achieve parallelism [4]. These self-migration processes synchronize among themselves using signal and wait events when they are on the same node. Studies using numerical algorithms have shown that NavP is as scalable as MPI (message passing interface) and as easy to use as DSM (distributed shared memory).

Previous research on NavP has been carried out using MESSENGERS developed at UC Irvine, which supports the C programming language only. While using C validated NavP’s high performance and ease of use, extending to supporting multiple programming languages across the board is key to taking it to the next level. We will leverage the DMTCP/python integration. Currently DMTCP provides a python module which supports application-initiated checkpointing using dmtcp.checkpoint() and dmtcp.restore(). We will build on top of the dmtcp module by adding a new DMTCP plugin dmtcp.hop(dest). A plugin enables one to modify the behavior of DMTCP, for example, by taking an additional action of moving the checkpoint memory image to the destination after checkpointing, and resume the computation there.

**Deliverable**: This proposal has three deliverables: (1) A software system and a procedure to achieve fault tolerance on NASA Pleiades using JPL Interferogram Generation (topsapp) as a test case. topsapp takes 3-5 hours to generate an interferogram; (2) NavP that supports python. We will develop NavP application prototypes using numerical algorithms or satellite data co-location algorithms; and (3) Using the NavP package and the prototypes, we will actively develop new proposals and seek collaboration and further funding at JPL as well as NASA levels.

**Innovation**: Fault tolerance is achieved from adapting DMTCP. The true innovation lies in NavP (1) No restructuring of the sequential algorithms in programming distributed and parallel computations, making apps easy to develop and maintain; (2) It enables SDS to address apps that are non-trivial to parallelize using distributed mobile pipelines; and (3) What’s being deployed in NavP is (virtual) infrastructure (i.e., the destination nodes in the Cloud). Computations are “deployed” dynamically through self-migration controlled by app programmers. This is in sharp contrast with the current approach to deploy computations using containers or web services. Not only does dynamic computation deployment by programmers make the Cloud as easy to program as desktop computers, but it also provides a lightweight solution, because what’s being moved is the app itself but not the run-time environment with hundreds of thousands of OS level files in a Docker image. NavP represents an innovative view -- the Lagrangian view in which the description of computation follows its locus migration to where the large data is. In contrast, all existing paradigms use the Eurlerian view in which computations are stationary and programs are thus broken into server and client segments that use messages or download/upload to facilitate communication [3].

**Significance**: This proposed work serves as a stepping-stone towards using an innovative view to building new SDS that is lightweight, efficient and effective, and easy to use and control by scientists and application programmers, and applying the new solution to building various workflows for different applications in the years to come.

**Milestones**: This proposed project will have three major milestones:

1. Adapting DMTCP. We are going to focus on adapting DMTCP to one of the two real projects: interferogram generation for ARIA HEC or satellite data collocation for WVCC. The goal is to familiarize ourselves with how DMTCP works and apply it to standalone runs of one of the above PGEs (product generation engine) with the Singularity container on NASA Pleiades supercomputer. A potential outcome of this milestone is for ARIA HEC or WVCC to officially support the integration of DMTCP with HySDS. This milestone will roughly take two calendar months from October to December of 2020;
2. Enabling NavP hop() statement using DMTCP plugin in python and demonstrating with simple algorithms how distributed programming is done using NavP. This is the core development effort of the entire proposed work. As prototype NavP programs in python are becoming available, we are going to collect performance data and other metrics in preparation for proposal and collaboration development. This milestone will roughly take five calendar months from January to May of 2021;
3. Parallel distributed mobile pipeline development for sophisticated numerical algorithms and proposal development. We will enable inter-process synchronization using Linux events to facilitate mobile pipelines, and parallelize numerical matrix algorithms that are known to be hard to parallelize. We will leverage the new NavP system developed and tested in the first seven months and the prototype apps along with their performance metrics to actively seek out collaborations and develop proposals that target to, e.g., JPL R&TD and NASA ROSES Funding Opportunities. This milestone will roughly take five calendar months from May to September of 2021.

**References**:

1. Jason Ansel, Kapil Arya and Gene Cooperman, DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop, *23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS'09)*, Rome, Italy, May, 2009.
2. Lei Pan, Lubomir F. Bic, and Michael B. Dillencourt, Distributed Sequential Computing Using Mobile Code: Moving Computation to Data, *International Conference on Parallel Processing, (ICPP’2001)*, Valencia, Spain, Spain, September, 2001.
3. Lei Pan, Ming Kin Lai, Koji Noguchi, Javid J. Huseynov, Lubomir F. Bic and Michael B. Dillencourt, Distributed Parallel Computing Using Navigational Programming, *International Journal of Parallel Programming*, Vol. 32, No. 1, Feb. 2004.
4. Lei Pan, Wenhui Zhang, Arthur Asuncion, Ming Kin Lai, Michael B. Dillencourt, and Lubomir F. Bic, Incremental Parallelization Using Navigational Programming: A Case Study, *International Conference on Parallel Processing (ICPP’2005)*, Oslo, Norway, June, 2005.

-------------------------------

when in the code and where on the node the computation and data rendewo

With web services or docker singularity based virtualization there is always a deployment stage when sds developers are involved. Navp deploys only facilities and scientific program developers are provided with utilities. New changes in the algorithm does not need to be deployed, the computation self migrate to

No server side code, no docker image build pipeline to run

Program a desktop vs program the cloud or cluster, no difference

Uses include: Fault tolerance (roll back to a previous checkpoint if a process crashes); extended sessions (restart the next time after a break); debugging (if a bug shows up restart from the last checkpoint under a debugger); reproducible bug reports (submit the last checkpoint from 30 seconds prior to a crash); fast startup of a process (checkpoint after the long initialization of a process)

DMTCP works on multi-threaded applications, multi-processes, one can check point computations on a cluster. Can check point matlab, R, python on Debian/Ubuntu, Fedora, and OpenSUSE. So it is an ideal vehicle to facilitate NavP going from a C prototype to real world applications.

DMTCP has a coordinator, DMTCP library is preloaded along with the user process startup, the DMTCP library starts a checkpoint thread and connects it to the coordinator. Before checkpointing, the checkpoint thread sends a signal to the user thread and blocks the user thread. The checkpointing thread and the user thread are not active at the same time.

principles: one DMTCP coordinator per user; the checkpoint thread and the user thread take turns to be active -- they are not active at the same time; as long as the checkpoint image is backed up, even if the coordinator dies, one can still restart from last checkpoint; runtime libraries are checkpointed into the memory image; the shell environment variables are in the memory image; no root/admin privileges are needed, all user-space

DMTCP supports OpenSHMEM, which will be used for NavP’s “node variables.” An example of this is a large matrix decomposed and distributedly stored on multiple nodes in the OpenSHMEM system.