CCNCheck: Enabling Checkpointed Distributed Applications in Content Centric Networks

Nitinder Mohan and Pushpendra Singh

Indraprastha Institute of Information Technology (IIIT), New Delhi, India

***Abstract*—We consider the problem of checkpointing a distributed application efficiently in Content Centric Networks so that it can withstand transient failures. We present CCNCheck, a system which enables a sender optimized way of checkpointing distributed applications in CCN's and provides an efficient mechanism for failure recovery in such applications. CCNCheck's checkpointing mechanism is a fork of DMTCP repository CCNCheck is capable of running any distributed application written in C/C++ language.**

Motivation

As CCN offers receiver-driven mode of communication, the distributed applications running on it needs to be modified from their usual sender-driven paradigm [1]. Checkpointing and rollback-recovery are well known techniques that allow processes to make progress in spite of failures [2]. However, CCN is devoid of any such mechanism of failure recovery. Keeping the above points in mind, we bring CCNCheck which offers a sender-optimized way of running distributed applications. CCNCheck also implements checkpointing for applications running on CCN.

Our Contribution

A typical distributed application in CCN is assumed to be running on multiple nodes and uses a common channel to send/receive interests and data. We further assume that:

1) Processes do not have any common clock/ memory.
2) Processes follow a fail-stop model of failing i.e. processes can crash by stopping execution and remain halted until restarted.

Checkpoint is saved local state of a process. Set of local states and messages in common channel is global state of a system. In checkpointing, every process takes a local checkpoint to ensure a global consistent state which can later be used to recover a system from failure [3].

Our work is centered around using interests as notifications/signals in CCN. Formally, a distributed system running on CCNCheck works on following model:

1) Every node knows about other nodes running the same distributed application.
2) Every node is defined by a unique name which is pre-appended by the application name the process is running.
3) The interest packet is not stored in the router's cache. The router only forwards the interest using its FIB entry.

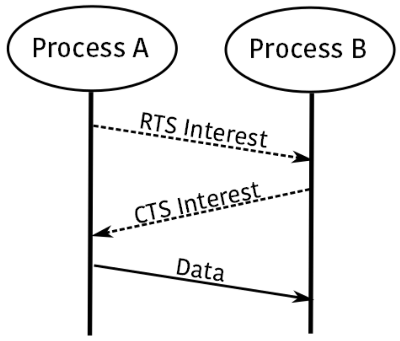

Figure 1: CCNCheck Communication Handler

A. Running Distributed Application

To run a sender-driven distributed application in CCN we use an approach very similar to solving hidden terminal problem in IEEE 802.11 networks. The sender first issues a **Request-to-Send (RTS)** interest to the desired destination process. This RTS packet acts as a notification to destination about an incoming data. The name of RTS contains the identifying name of its issuer using which the destination process issues a **Clear-to-Send (CTS)** interest back to the sender. CTS serves as the necessary interest required to send the data in CCN. Figure 1 depicts process A sending some data to process B using CCNCheck.

B. Enabling Checkpoint

Distributed Multi-Threaded Checkpoint (DMTCP) is a research-based, transparent, user-level checkpointing tool for distributed applications. DMTCP follows a blocking type algorithm of checkpointing to ensure a global consistent state at each checkpoint. It employs a stateless centralized coordinator to coordinate checkpoint requests between nodes [4].

We have developed a plugin for DMTCP which enables it to work in CCN environment. Even though CCN is deployed as an overlay on TCP/IP networks [5] for which DMTCP works well, however, some more logical changes are necessary to make DMTCP function in CCN. We also formalize various inconsistent checkpoint scenarios due to uncoordinated checkpointing in CCN and devise a method to overcome such situations. Some of the changes made are:

1) DMTCP uses flush token to clear out TCP sockets during checkpoint process to ensure consistent checkpoint. As CCN works atop of Interests and Data packets, we have designed a "Flush Interest" which ensures that checkpoint is consistent from any orphan interests and data.
2) DMTCP coordinator is modified to detect a CCN network and register itself with CCN Daemon on invoking.
3) The Coordinator is run as a stateless process with a name unique to the environment/organization. We have designed interest packets which is used by coordinator to checkpoint processes in the application.
4) The restart from checkpoint process is able to resolve any non-responded interests due to lack of Pending Interest Table (PIT) entries.
5) The discovery services in reconnect phase on restart from a checkpoint works using CCN namespaces.

IMPLEMENTATION

I System Model

CCNCheck uses three layer abstraction model.

a) **Communication Handler:** It handles the Interest and Data packets to be sent between communicating nodes. It is built on CCNx v0.8.2.
b) **Checkpoint Handler:** It provides the checkpoint mechanism in CCN and is based on DMTCP.
c) **End-User Applications:** These are applications to be run in a distributed environment. It can be in C/C++ language.

II Interest Naming Rules

The naming format for RTS and CTS packets in CCNx are as follows:

ccnx://Application Name/Receiver Address/Type of Interest/Sender Address

The *Request-To-Send* and *Clear-To-Send* interests use signal name ’RTS’ and ’CTS’ respectively. The checkpoint interest, however, is only one-way notification (i.e. from coordinator to process). Thus, interest name does not have the sender’s name appended to it and it is denoted by signal type ’check’. Similarly, *Flush Interest* has a signal name ’flush’ but is appended with the last name of last interest sent Figure 2 shows naming rules for CCNCheck.

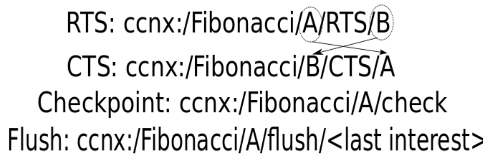
RTS: ccnx:/Fibonacci/A/RTS/B
CTS: ccnx:/Fibonacci/B/CTS/A
Checkpoint: ccnx:/Fibonacci/A/check
Flush: ccnx:/Fibonacci/A/flush/<last interest>

Figure 2: Naming Rules in CCNCheck

III Applications

CCNCheck was deployed on a test-bed of six interconnected nodes in CCN network. We have developed two sample distributed applications to review our system. We have also used an existing application to check the compatibility of our system.

a) A simple C application which keeps counting till infinity is run locally on each node with different start times and is killed later. The goal was to check the consistency of checkpoint taken by CCNCheck before failure.
b) A distributed C++ application in which the participating nodes compute the consecutive numbers of fibonacci sequence in an iterative manner. This application utilizes the distributed capabilities of CCNCheck to send the result to the next node after each subsequent computation.
c) A CCN enabled VLC player which can stream videos on a Content Centric Network.

We are able to checkpoint all the applications listed above.

REFERENCES

[1] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, “Voccn: voice-over content-centric networks,” in *Proceedings of the 2009 workshop on Re-architecting the internet*. ACM, 2009, pp. 1–6.

[2] R. Koo and S. Toueg, “Checkpointing and rollback-recovery for distributed systems,” *Software Engineering, IEEE Transactions on*, no. 1, pp. 23–31, 1987.

[3] K. M. Chandy and L. Lamport, “Distributed snapshots: determining global states of distributed systems,” *ACM Transactions on Computer Systems (TOCS)*, vol. 3, no. 1, pp. 63–75, 1985.

[4] J. Ansel, K. Arya, and G. Cooperman, “DMTCP: Transparent checkpointing for cluster computations and the desktop,” in *23rd IEEE International Parallel and Distributed Processing Symposium*, Rome, Italy, May 2009.

[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, “Networking named content,” in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.