CTK: A Consensus Toolkit

Course Project: CSE 223A (Principles of Distributed Computing)
Winter 2004

Siddhartha Saha

1. Introduction:

This report describes CTK (*A Consensus ToolKit*), which is a framework for simulating and evaluating various kinds of consensus protocols. A brief overview of this report is as follows:
The next section [Sec 2] provides an overview of the system architecture and framework. Section 3 provides somewhat detailed description of the interface provided by the framework and what is the user required to do in order to use this framework to design any consensus protocol and test it. This section is divided into various subsections as required. The next section [Sec 4] provides a brief overview of the three consensus protocols that are implemented using this framework.

2. Overview

The toolkit framework that we have developed for this project is build and tested on Visual C++ .Net. Since, one of the goals of this project was to examine the consensus protocols over different network topologies, and it was required that the user should be able to tweak the underlying network parameters as required, the simulator framework is built on a single threaded event driven architecture. Messages are sent and received using event driven callback methods. A central scheduler class is used to maintain the event driven architecture of the framework. All messages in the system are scheduled in the scheduler and dispatched by the scheduler.

The framework supports three kinds of underlying network topologies over which any consensus protocol may be run and tested. The first one is the trivial network, where all the processes share a central shared medium, where any message delivered to the channel can be received by all the processes. The second type of network is the broadcast bus type network where every process is connected to R links and the links are connected to R channels. The third type of network supported is a custom graph kind of network. This provides the ultimate flexibility to the user. The user can explicitly describe the graph to the simulator. For the consensus protocol, the user is required to provide a C++ class which contains the implementation of the protocol. The details are given in the next section.

3. Detailed Description:

3.1 Important Classes:

- **CTKScheduler:**
  - *Implementation file:*
    - CTKScheduler.cpp, CTKScheduler.h
  - *Exported Interface:*
    - `static CTKScheduler * Instance()`: Returns a pointer to the instance of the CTKScheduler class.
    - `void Schedule(CTKEvent * event, double delay)`: Schedules an event to run after '*delay*' seconds.
    - `double Clock()`: return the value of the current system virtual clock. The clock information is maintained by the scheduler.
    - `int DispatchEvent()`: This function is called by the framework to dispatch an event from the scheduler. User will never need to call this function.
  - *Importance for the user:*
    - This class is of no direct use to the user implementing the protocol. But in some protocols, if the user needs to register some callback events for the use of the protocol, the `Schedule` function may be used. Also, if the system time is needed to know at some point, the `Clock` function may be used. A pointer to an instance to the CTKScheduler class can be obtained by calling `CTKScheduler::Instance()`.
- **CTKEvent:**
  - *Implementation files:*
    - CTKEvent.cpp, CTKEvent.h
  - *Importance to the user:*
    - This is fairly important as the protocol will be using this event class in its Handler method. The user needs to extract the payload data from this event data, and act accordingly. Various system failure messages also will come encapsulated in this event data. So, when a process receives a message `MSG_CRASH`, it should update its status to crash.
  - *Exported Interface:*
    - `CTKEventHandler * handler` : The entity/class which will handle the event. This class must be inherited from the class CTKEventHandler, and must have an implementation of the `Handle(CTKEvent * event)` method.
    - `CTKEventHandler * generator` : The entity which generated this event.
    - `double time` : The time at which the event will be handled. The user should never edit this value.
    - A `PAYLOAD` structure, which is defined in CTKEvent.h

        - The `PAYLOAD.data` is the user created data that will be passed between processes. The `PAYLOAD.type` is an integer value which denotes what kind of message or event is. The framework notifies the process various events through this `PAYLOAD.type` field. For system generated message the field will be any of these: { `MSG_FAIL_CRASH, MSG_FAIL_BYZ, MSG_FAIL_SEND_OMISSION, MSG_FAIL_RECV_OMISSION, MSG_FAIL_OMISSION, MSG_ROUND_BEGIN}`

- **CTKLink:**
    - *Implementation files:*
        - CTKLink.cpp and CTKLink.h
    - *Importance to the user:*
        - The user protocol should not use this class directly. This class is mentioned here just for completeness of information. This is the base class which the framework uses to implement the connection between processes.
- **CTKSysModel:**
    - *Implementation files:*
        - CTKSysModel.cpp and CTKSysModel.h
    - *Description:*
        - This class is the main class which sets up the framework and underlying network for the simulation. The user needs to initialize this class before running the simulation and then call the Run method of this class to start the simulation.
    - *Exported Interface:*
        - `int InitializeSingle(int nProc , double dMaxDelay ,int nMaxFail, CTKProcess ** procarray, int round, double rountTIme, int procFailType = FAILMODEL_CRASH)` : Function to initialize the system model with a trivial network where all nodes are connected to every other node.
        - `int InitializeBBus (int nproc ,int nchannel ,double delay ,int procfail , int chanfail ,int linkfail,CTKProcess ** procarray ,int nRound ,double dRoundTime, int procFailType = FAILMODEL_CRASH, int linkFailType = FAILMODEL_CRASH)`: Function to initialize the system model for a broadcast bus type of network
        - `int InitializeGraph (char * graphFile, int procfail, int linkfail, CTKProcess ** procarray ,int nRound ,double dRoundTime, int procFailType = FAILMODEL_CRASH, int linkFailType = FAILMODEL_CRASH)`: Function to initialize the system model for a customized graph.
        - `void Run(double StartTime)`: After initialization, this call runs the simulation.
    - *Importance to the user:*
        - This is an important class, since the user need to instantiate this class, initialize it properly and call the Run() method to runs the simulation.

- **CTKProcess:**
  - *Implementaion Files:*
    - CTKProcess.cpp and CTKProcess.h
  - *Description:*
    - This is the most important class of the lot, since the protocol implementation works by making a class which directly inherits from this class.
  - *Useful Interface and Variables:*
    - `void Decide(int v)` : This function is used to when the process wants to decide on a particular value v.
    - `void Init()`: Implement this function
    - `int SendToAll(void * data, int type)`: Send Primitive to use when sending some message to all processes. `type` is the type of message to distinguish the message from framework messages.
    - `int Send(void * data, int index, int type)` : Send message to a process with index '`index`'. This is to be used when using a custom graph underlying network. The framework will forward the message to the next hop along the shortest path towards destination. User is responsible for forwarding the message towards its final destination by calling this function at intermediate nodes.
    - `int SendToLink(void * data, int link, int type)`: Send message to a particular link, to be used when using the broadcast bus kind of network.
    - `int nLinks` : Number of links connected to this process.
    - `CTKLink ** ptrLinks` : Array of pointers to links that are connected to this process. This array may be used to determine from which link the current message event has come, by comparing the generator field of the payload of the event to the entries of this array.
    - `int nProcesses` : Number of processes in the system.
    - `int nMaxFail`: Number of maximum processes that may fail.
    - `int iProcIndex`: Index of the current process

3.2 Steps towards Implementing a Protocol:

1. Inherit a class from the CTKProcess class.
2. Implement the function to initialize all the protocol dependent data: `void Init()`
3. Implement the `void Handle(CTKEvent * event)` function which should properly implement the protocol. The various methods and data available to the process in listed above.
4. Check for framework sent messages like the message to indicate a new round is beginning () and the crash related messages (). For the crash related messages, it is the users responsibility to act properly, i.e., to, crash the system. For `MSG_FAIL_CRASH` the system should not process any more messages. For

`MSG_FAIL_RECV_OMISSION`, the process may send messages, but will be unable to receive any message from other process. For `MSG_FAIL_SEND_OMISSION`, the process may receive messages but will not be able to send message. For `MSG_FAIL_BYZ` the process may show arbitrary behavior. What kind of arbitrary behavior a process shows is dependent on the protocol, so, coding that is the user's responsibility.

3.3 Steps towards Running a Protocol Simulation:

1. Implement a protocol and decide on the underlying network model.
2. Instantiate a CTKScheduler class.
3. Instantiate a CTKSysModel class, and initialize it with the corresponding initialization function depending on the underlying network model.
4. Call the Run(..) method of the CTKSysModel class.
5. Compile the code and run.

4. Protocols Implemented using this CTK Framework:

5.1 t + 1 Round Based Protocol for crash failures: This protocol is implemented in the files `ProcCrashFailure.h` and `ProcCrashFailure.cpp.`

5.2 Consensus with broadcast buses: Process may Send Omission: This protocol is implemented in the files `ProcBBusSendOmission.h` and `ProcBBusSendOmission.cpp.`

5.3 Consensus with broadcast buses: Process may Arbitrary failure: This protocol is implemented in the files `ProcBBusByzantine.h` and `ProcBBusByzantine.cpp.`

The main driver file, containg the main function is the `ctk.cpp` file.

5. Appendix:

Graph File format:

<integer - N> - Number of process
<integer - E> - Number of edges
In next E lines, place the two nodes [0 based index], and the max delay bound of the link

<integer> <integer> <double>
<integer> <integer> <double>
.
. (E Lines)
.
<integer> <integer> <double>

Typical Run:

```
Scheduled fails: Proc: 9, Channel: 1, Link: 3
Scheduler Count: 13
[0.461440] Channel [ID = 66] Fails.
[1.074252] Process 17 enters byzantine state.
[1.637928] Process 12 enters byzantine state.
[2.784204] Process 0 enters byzantine state.
[4.059877] Process 19 enters byzantine state.
[4.928434] Process 23 enters byzantine state.
[5.068514] Channel [ID = 51] Fails.
[5.897702] Process 14 enters byzantine state.
[5.970641] Channel [ID = 7] Fails.
[6.035646] Process 3 enters byzantine state.
[7.250587] Process 21 enters byzantine state.
[8.786279] Channel [ID = 93] Fails.
[9.340800] Process 7 enters byzantine state.
[30.000000] Process 0 deciding on 20 [Faulty Process]
[30.000000] Process 1 deciding on 20
[30.000000] Process 2 deciding on 20
[30.000000] Process 3 deciding on 20 [Faulty Process]
[30.000000] Process 4 deciding on 20
[30.000000] Process 5 deciding on 20
[30.000000] Process 6 deciding on 20
[30.000000] Process 7 deciding on 20 [Faulty Process]
[30.000000] Process 8 deciding on 20
[30.000000] Process 9 deciding on 20
[30.000000] Process 10 deciding on 20
[30.000000] Process 11 deciding on 20
[30.000000] Process 12 deciding on 20 [Faulty Process]
[30.000000] Process 13 deciding on 20
[30.000000] Process 14 deciding on 20 [Faulty Process]
[30.000000] Process 15 deciding on 20
[30.000000] Process 16 deciding on 20
[30.000000] Process 17 deciding on 20 [Faulty Process]
[30.000000] Process 18 deciding on 20
[30.000000] Process 19 deciding on 20 [Faulty Process]
[30.000000] Process 20 deciding on 20
[30.000000] Process 21 deciding on 20 [Faulty Process]
[30.000000] Process 22 deciding on 20
[30.000000] Process 23 deciding on 20 [Faulty Process]
    ------  RESULT  --------
 Total number of processes: 24
 Total number of failed process: 9
 Total number of decided Non Faulty processes: 15
 Number of rounds: 3
 Consensus Reached: YES
```

Byzantine Failure with broadcast bus

```
Scheduled fails: 16
[10.948210] Process 3 Crashes
[11.702628] Process 14 Crashes
[16.401257] Process 11 Crashes
[17.391888] Process 10 Crashes
[18.034608] Process 2 Crashes
[20.000000] Process 0 deciding on 100
[23.000000] Process 1 deciding on 100
[23.000000] Process 4 deciding on 100
[23.000000] Process 5 deciding on 100
[23.000000] Process 6 deciding on 100
[23.000000] Process 7 deciding on 100
[23.000000] Process 8 deciding on 100
[23.000000] Process 9 deciding on 100
[23.000000] Process 12 deciding on 100
[23.000000] Process 13 deciding on 100
[23.000000] Process 15 deciding on 100
[23.000000] Process 16 deciding on 100
[23.000000] Process 17 deciding on 100
[23.000000] Process 18 deciding on 100
[23.000000] Process 19 deciding on 100
    ------  RESULT  --------
 Total number of processes: 20
 Total number of failed process: 5
 Total number of decided Non Faulty processes: 15
 Number of rounds: 21
 Consensus Reached: YES
```

Crash Failure

```
Scheduled fails: Proc: 2, Channel: 0, Link: 4
 [0.694357] Channel [ID = 191] Fails.
[1.049226] Process 4 Send Omission Fails.
[5.071078] Process 10 Send Omission Fails.
[6.026551] Channel [ID = 96] Fails.
[8.297128] Channel [ID = 130] Fails.
[11.719474] Channel [ID = 160] Fails.
[20.000000] Process 0 deciding on 100
[26.000000] Process 1 deciding on 100
[26.000000] Process 2 deciding on 100
[26.000000] Process 3 deciding on 100
[26.000000] Process 4 deciding on 100 [Faulty Process]
[26.000000] Process 5 deciding on 100
[26.000000] Process 6 deciding on 100
[26.000000] Process 7 deciding on 100
[26.000000] Process 8 deciding on 100
[26.000000] Process 9 deciding on 100
[26.000000] Process 10 deciding on 100 [Faulty Process]
[26.000000] Process 11 deciding on 100
[26.000000] Process 12 deciding on 100
[26.000000] Process 13 deciding on 100
[26.000000] Process 14 deciding on 100
[26.000000] Process 15 deciding on 100
[26.000000] Process 16 deciding on 100
[26.000000] Process 17 deciding on 100
[26.000000] Process 18 deciding on 100
[26.000000] Process 19 deciding on 100
    ------  RESULT  --------
 Total number of processes: 20
 Total number of failed process: 2
 Total number of decided Non Faulty processes: 18
 Number of rounds: 3
 Consensus Reached: YES
```

Send omission failure with broadcast bus