Extending IncrediBuild To Linux

Phase I - POC

Acceptance Test Procedure (ATP)

Draft

Author:

Dmitry Kuzminov

Approved:

Dori Exterman

Version:

2.0

Table of Contents

Introduction

The purpose of this document is to define methods and criteria to evaluate the POC stage of the Xoreax Incredibuild for Linux project.
The following aspects are to be tested by the POC implementation:

1. Scalability
2. Cache
3. Virtualization
4. Process output (piping -> stdOut, stdErr)
5. Parallelization of scripts tasks (like XgSubmit does)
6. Enable to re-play the ATP on-site

The evaluation serves two purposes:

1. Determine whether the implementation correctly utilized/implemented the technological feature. (evaluation of the contractor work)
2. Determine whether the implementation matches the requirements of the final product. (input for further research/design efforts)

Scope

The following groups of tests are currently identified for POC test:

1. Parallel execution of multiple processes.
2. Remote execution of processes, virtualization[1] support.
3. I/O pipes to remote processes (stdout/stderr), logging of output.
4. Cache mechanisms of data and execution environment on the remote execution platforms.

1 Virtualization refers to local cloning of the execution environment on the initiating host and not a virtual machine.

Tested product description

1. There will be a script that will execute commands using a tool that will simulate XgSubmit. For example:
    1.1. XgSubmit [process a] param1 [Process to be executed, parameters to process]
    1.2. XgSubmit [process a] param2
2. Process [process a] will execute a process that will be passed as the first parameter and pass the parameters that are the second parameter
    2.1. The process to be executed, for example [process b] will not be available on the remote machine and will need to be virtualized
    2.2. Process A and B will also use a library that only exists on the local machine
    2.3. [process b] will receive the following parameters:
        2.3.1. Input file (large, 50 mega)
            2.3.1.1. We expect to see that after the first time this file was read (transferred from the local machine) the next reads will be faster due to caching
            2.3.1.2. We expect that after doing some other stuff on that machine and doing another run of this ATP test, the input file will be read fast again (will still be cached on the remote machine) if it didn't change on the local machine
            2.3.1.3. We expect that in the 3rd execution of this test, once we will change something in the input file on the local machine, the helper machine will read the changed file and not the cached one and will cache the new file again
            2.3.1.4. We expect that once this Helper machine will act as an Initiator and then will act as a Helper machine for this process again, the input file will be read fast again (will still be cached on the remote machine) if it didn't change on the local machine
            2.3.1.5. We expect that if process b will be executed with a different input file, both files will be cached.

2.3.1.6 We expect that the cache will be able to contain at least 2G of file capacity (Dimitry will find out NFS cache limitations)

2.3.1.7 We expect that there will be no limitation to the size of a single file that can be cached (the poc will once test a 2G file)

2.3.1.8 We expect that once there will be no more space in the cache (cache size can be a constant value), the older files will be removed from the cache and new files will be cached

2.3.2 Number of read times

2.3.3 stdErr message that if exists will generate an exit code 5

2.4 Process [process b] will write two output files:

2.4.1 The input file copied as an output file

2.4.1.1 The Input file will be read entirely and written to a new file (there will not be a copy file operation)

2.4.2 A log file with the same text as the stdOut

2.5 Process [process b] will write to stdOut all the operations that it does (every line will have a timestamp): (Note, Linux implementation does not have a layer of virtualization between the process and the OS so everything runs without any “code hooks”. The info below can be either outputted by the process logic itself or a p-trace tool)

2.5.1 Start

2.5.2 Before loading the dll

2.5.3 Start synching the Dll from the initiating machine

2.5.4 End synching

2.5.5 Loading the Dll

2.5.6 Before opening the input file

2.5.7 Start synching the input file

2.5.8 End synching

2.5.9 Reading file X… (as many times as the read parameter indicate)

2.5.10 Finished reading

2.5.11 Start writing

2.5.12 End writing

2.5.13 Exiting

3 Process [process a] will forward all the[process b] process outputs to the script

4 When the script will run without XgSubmit command, all tasks will be executed serially, locally, no IB no intervention

5 The XgSubmit command will make all the tasks run in parallel

6 The script will echo all the outputs received from process [process a] (including the error message and code)

7 This ATP will be executed using 50 EC2 Linux machines

8 Process a and process b will be written by Galil soft and will be reviewed by Xoreax as a first step.

Multiple process execution

Test 00001 – tasks are queued on local machine and executed on remote machine, stdOut returned back to initiator machine

1 **XgSubmit** push process a to **Server** queue

1.1 **Server** write **log -** “running index, timestamp, process [a] pushed to queue”

2 **Server** pop process a from queue and execute it in **BuildHelper** on remote machine

2.1 **Server** write **log -** “running index, timestamp, process a is executed”

2.2 Process a will write **stdOut -** “ timestamp, process is running”

3 Process a will execute process b

3.1 Process a write **stdOut -** “timestamp, process b executed”

3.2 process a will continue working, but will wait process b to be finished

3.3 **Server** will not queue process b (b would be executed directly from a)

3.4 process b will write **stdOut -** “timestamp, process is running”

4 b will not execute any other process, it will process file copy (by reading a file and writing what’s read to another file) and exit

4.1 Process b will write **stdOut -** “timestamp, start file copy”

4.2 Process b will write **stdOut -** “timestamp, finished file copy”

4.3 Process b will write **stdOut -** “timestamp, process exit”

4.4 Process a will write **stdOut -** “timestamp, process exit”

5 expected results

5.1 submit 100 tasks

5.2 log file shows tasks a > 10 started before first task a is finished (parallel execution) - meaning at least 10 processes have been started in parallel simultaiously.

5.3 log file shows on what machine process a executed

5.4 log file shows for each process a index and mark for each index finish time

5.5 log file shows successful execution of processes a and b

5.6 StdOut, StdErr of process a and b will be written in the Initiating machine

*Test 00002 – tasks are queued on local machine and executed on remote machine. **stdErr** is returned back to initiator machine*

1 **XgSubmit** push process a to **Server** queue (Build System, Daemon)

1.1 **Server** write **log**- “running index, timestamp, process [a] pushed to queue”

2 **Server** pop process a from queue and execute it in **BuildHelper** on remote machine

2.1 **Server** write **log**- “running index, timestamp, process a is executed”

2.2 Process a will write **stdOut**- “ timestamp, process is running”

3 Process a will execute process b

3.1 Process a write **stdOut**- “timestamp, process b executed”

3.2 process a will continue working, but will wait for process b to be finished

3.3 **Server** will not queue process b (b would be executed directly from a)

3.4 process b will write **stdOut**- “timestamp, process is running”

4 b will not execute any other process, it will process file copy and generate segmentation fault

4.1 Process b will write **stdOut**- “timestamp, start file copy”

4.2 Process b will write **stdOut**- “timestamp, finished file copy”

4.3 Process b will write **stdErr** - “segmentation fault” (message generated by system, could be other format)

4.4 Process b will not write **stdOut**- “timestamp, process exit”

4.5 Process a will write **stdOut**- “timestamp, process exit” or will return **stdErr**

5 expected results

5.1 submit 100 tasks

5.2 receive back stdErr from processes

5.3 identify failed process

5.3.1 process name

5.3.2 execution machine

5.3.3 **stdErr** message

Remote execution supports Virtualization

Test 00003 – Initiator machine create Slot (Helper Machine) Virtualization on remote machine

1 **Server** open grid configuration file and create **Slots**

1.1 **Server** write **log -** “Slot index, machine name, ip address, Slot creation starts”

2 Slot creation

2.1 Open SSH session to remote machine

2.1.1 Slot write log - “Slot index, machine name, SSH opened”

2.2 NFS (File system virtualization)

2.2.1 /tmp/machine_disk (machine=initiator) doesn't exist

2.2.1.1 create folder /tmp/machine_disk

2.2.1.2 write log “Slot index, machine name, created /tmp/machine_disk”

2.2.2 /tmp/machine_disk is empty (There is a tree but it’s empty)

2.2.2.1 mount (connect) /tmp/machine_disk to initiator machine

2.2.2.2 write log “Slot index, machine name, NFS connected”

2.2.3 /tmp/machine_disk is not empty, /etc/machine_name doesn't exists

2.2.3.1 write log “Slot index, machine name, /tmp/machine_disk bad data”

2.2.3.2 remove /tmp/machine_disk

2.2.3.3 write log “Slot index, machine name, removing /tmp/machine_disk”

2.2.3.4 loop for n times to section 2.2

2.2.4 /tmp/machine_disk is correct

2.2.4.1 write log “Slot index, machine name, /tmp/machine_disk is ready for Virtualization”

2.3 Virtualization

2.3.1 chroot (Virtualization) on /tmp/machine_disk

2.3.2 write log “Slot index, machine name, Virtualization done”

3 expected results

3.1 100 slots created

3.1.1 connect to 50 machines

3.1.2 create 2 Slots for each machine

3.2 Close all Slots and reconnect

3.2.1 connect to 50 machines

3.2.2 create 2 slots for each machine

3.2.3 stages 2.2.1, 2.2.2 should be bypassed. Slots creation should be faster

3.2.3.1 remove on each computer /tmp/machine_disk folders

3.2.3.2 measure with statistics section 2.2.1, 2.2.2, 2.2.3, 2.2

Test 00004 – Initiator machine create Slot Virtualization with Cache mechanisms

1 **Server** open grid configuration file and create **Slots**

1.1 **Server** write **log** - “Slot index, machine name, ip address, Slot creation starts”

2 Slot creation

2.1 Open SSH session to remote machine

2.1.1 Slot write log - “Slot index, machine name, SSH opened”

2.2 NFS, CacheFS

2.2.1 /tmp/machine_disk (machine=initiator) doesn't exist

2.2.1.1 create folder /tmp/machine_disk

2.2.1.2 write log “Slot index, machine name, created /tmp/machine_disk”

2.2.2 /tmp/machine_disk is empty

2.2.2.1 mount (connect) /tmp/machine_disk to initiator machine

2.2.2.2 write log “Slot index, machine name, NFS connected”

2.2.2.3 CacheFS /tmp/machine_disk

2.2.2.4 write log “Slot index, machine name, /tmp/machine_disk is cached”

2.2.3 /tmp/machine_disk is not empty, /etc/machine_name doesn't exists

2.2.3.1 write log “Slot index, machine name, /tmp/machine_disk bad data”

2.2.3.2 remove /tmp/machine_disk

2.2.3.3 write log “Slot index, machine name, removing /tmp/machine_disk”

2.2.3.4 loop for n times to section 2.2

2.2.4 /tmp/machine_disk is correct

2.2.4.1 write log “Slot index, machine name, /tmp/machine_disk is ready for Virtualization”

2.3 Virtualization

2.3.1 chroot (Virtualization) on /tmp/machine_disk

2.3.2 write log “Slot index, machine name, Virtualization done”

3 expected results

3.1 100 slots created

3.1.1 connect to 50 machines

3.1.2 create 2 Slots for each machine

3.2 Close all Slots and reconnect

3.2.1 connect to 50 machines

3.2.2 create 2 slots for each machine

3.2.3 stages 2.2.1, 2.2.2 should be bypassed. Slots creation should be faster

3.2.3.1 remove on each computer /tmp/machine_disk folders

3.2.3.2 measure with statistics section 2.2.1, 2.2.2, 2.2.3, 2.2

Test 00005 – Initiator machine create Slot Virtualization with Union of local system and network file system – purpose of this test is to check whether we can use linux system libraries from the remote machine and not cache / NFS them

1 **Server** open grid configuration file and create **Slots**

1.1 **Server** write **log** - “Slot index, machine name, ip address, Slot creation starts”

2 Slot creation

2.1 Open SSH session to remote machine

2.1.1 Slot write log - “Slot index, machine name, SSH opened”

2.2 NFS, CacheFS, UnionFS

2.2.1 /tmp/machine_disk (machine=initiator) doesn't exist

2.2.1.1 create folder /tmp/machine_disk

2.2.1.2 write log “Slot index, machine name, created /tmp/machine_disk”

2.2.2 /tmp/machine_disk is empty

2.2.2.1 mount (connect) /tmp/machine_disk to initiator machine

2.2.2.2 write log “Slot index, machine name, NFS connected”

2.2.2.3 bind (connect) local system libraries to /disk/machine_disk

2.2.2.4 write log “Slot index, machine name, local system files used”

2.2.2.5 CacheFS /tmp/machine_disk

2.2.2.6 write log “Slot index, machine name, /tmp/machine_disk is cached”

2.2.3 /tmp/machine_disk is not empty, /etc/machine_name doesn't exists

2.2.3.1 write log “Slot index, machine name, /tmp/machine_disk bad data”

2.2.3.2 remove /tmp/machine_disk

2.2.3.3 write log “Slot index, machine name, removing /tmp/machine_disk”

2.2.3.4 loop for n times to section 2.2

2.2.4 /tmp/machine_disk is correct

2.2.4.1 write log “Slot index, machine name, /tmp/machine_disk is ready for Virtualization”

2.3 Virtualization

2.3.1 chroot (Virtualization) on /tmp/machine_disk

2.3.2 write log “Slot index, machine name, Virtualization done”

3 expected results

3.1 100 slots created

3.1.1 connect to 50 machines

3.1.2 create 2 Slots for each machine

3.2 Close all Slots and reconnect

3.2.1 connect to 50 machines

3.2.2 create 2 slots for each machine

3.2.3 stages 2.2.1, 2.2.2 should be bypassed. Slots creation should be faster

3.2.3.1 remove on each computer /tmp/machine_disk folders

3.2.3.2 measure with statistics section 2.2.1, 2.2.2, 2.2.3, 2.2

Remote execution collects correctly stdout/stderr

Test 00006 – tasks a, a', a'', b, b' will be used, stdOut/stdErr returned back to initiator machine and identify clearly errors

1. **XgSubmit**push processes a, a' and a'' to **Server**queue
    - 1.1 process a – is errorless
    - 1.2 Process a' – is executing process b'. process b' generates error
    - 1.3 process a'' – generate error
    - 1.4 **Server** write **log**- “running index, timestamp, process [a|a'|a''] pushed to queue”
2. **Server** pop process [a|a'|a''] from queue and end execute it in **BuildHelper** on remote machine
    - 2.1 **Server** write **log**- “running index, timestamp, process [a|a'|a''] is executed”
    - 2.2 Process a will write **stdOut**- “ timestamp, process is running”
3. Process [a|a'|a''] will execute process [b|b'']
    - 3.1 Process a write **stdOut**- “timestamp, process b executed”
    - 3.2 Process a' write stdOut - “timestamp, process b' executed”
    - 3.3 Process a'' will generate stdErr
4. **Server** will not queue process [b|b'] ([b|b'] would be executed directly from [a|a'|a''])
    - 4.1 process b will write **stdOut**- “timestamp, process is running”
    - 4.2 process b' will generate system error and use stdErr
5. Process b will not execute any other process, it will process file copy and generate segmentation fault
    - 5.1 Process b will write **stdOut**- “timestamp, start file copy”
    - 5.2 Process b will write **stdOut**- “timestamp, finished file copy”
    - 5.3 Process b will write **stdOut** - “timestamp, process exit”

6 Process b' will write **stdErr**- “segmentation fault” (message generated by system, could be other format)

7 expected results

- 7.1 submit 99 tasks (50 helpers)
  - 7.1.1 33 a processes
  - 7.1.2 33 a' processes
  - 7.1.3 33 a" processes
- 7.2 receive back stdErr from processes and identify failed ones
- 7.3 identification of failed process
  - 7.3.1 process name
  - 7.3.2 execution machine
  - 7.3.3 **stdErr** message
- 7.4 Should be 33 successful processes, 33 failed process a", 33 failed processes b'
  - 7.4.1 log should use names and indices for clear tracking of process

Cache mechanisms for Virtualization

Test 00007 – Slot Virtualization session cached big files

1. Clean /tmp/ folder on all machine in grid
2. Run test 00005 on machine A
3. Run test 00005 on machine B
4. Run test 00005 on machine A
5. Will be executed in a kind sequence (will benchmark 5 machines, then 10, 20 ,50)
6. expected results
    - 6.1 test 00005 finished successful on machine A and machine B
    - 6.2 section 3 of test 00007 shows clearly faster result
        - 6.2.1 measure time to copy large files
        - 6.2.2 identify all processes, slots and initiator machines
        - 6.2.3 generate table of comparison

Note: Network drives

Initial research on the ability to work with network drives