EnduroX System Overview

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1.0	2012-11	Initial draft	MV

Contents

Chapter 1

What is EnduroX?

EnduroX is middle-ware software that composes semantics of Application Transaction Monitor Interface (ATMI). Also EnduroX provides UBF (Universal Buffer Format) so programmers can use self described buffers for inter-process communication purposes.

System principles have similarities with Oracle ® Tuxedo ® system.

EnduroX consists release consists of various components. Like libraries for C/C++ programs, transaction monitor it self, and command line administration utility.

On each machine transaction monitor process runs locally, it controls so called server processes, which are programs which are started by transaction monitor and they advertise services. It is possible to join together many transaction monitor processes under same machine or over multiple machines. This makes a cluster.

In cluster environment services are shared over the linked local transaction monitors. It is possible to load balance some service which runs on local machine and on some other machine. The caller/client might be unaware on which machine the call actually is served.

Chapter 2

How EnduroX Differ from Oracle® Tuxedo®

Basically EnduroX is consider as clone of Tuxedo. EnduroX have source level compatibility with Tuxedo. However there are set of functionality which is missing in EnduroX. This is the list what EnduroX does NOT support:

1. No support for FML views;
2. No FML indexing;
3. Some FML functionality is not implemented (but mostly all is supported);

Improvements over Tuxedo:

1. ATMI servers can register custom poll file descriptors with own callback function;
2. ATMI servers can register periodic callback functions while ATMI server is idling;
3. Cluster is peer based, there is no master node;
4. Much simpler configuration reload during running app. Just edit *ndrxconf.xml* and run *xadmin reload* command. Tuxedo UBB configuration file is much harder to update in running up.
5. System can run with out the main Enduro/X daemon (ndrxd). It is much like running the Tuxedo with out Bulletin Board, which is not possible for Tuxedo. Even if *ndrxd* daemon dies for some reason, it is possible to start in back in running application server instance with out service interruption.
6. Simple TMQUEUE setup. Allows run-time configuration updates and defaulted queue setups.

Chapter 3

What EnduroX deliverable contains?

The distribution package contains following list of resources:

Table 3.1: Distribution package description - "bin" directory

Resource	Description
buildclient	Wrapper script for client process build. Basically it links binary with required EnduroX libraries
buildserver	Wrapper script for building server executables. Basically it links developers object files with libatmisrv.so which contains main() entry of the process
fdown.sh	Internal shell script forced shutdown of the transaction monitor. Part of 'ndrx' admin utility
mkfldhdr	Generates C header files for UBF buffers
ndrxd	EnduroX deamon. This process is central point of the application which does the Application Transaction Monitoring
tmadmin	Alias for xadmin
tpbridge	Special ATMI server which drives the connectivity for cluster purposes. It allows to connect one EnduroX instance to another EnduroX instance
tpevsrv	Special ATMI server which serves the local even distribution over the services
ud	ATMI client utility. This reads stdin of dumped UBF buffer. And sends request to specified service. when responses is received, it is dumped back to stdout.
xadmin	EnduroX Administration command utility. Basically utility is also responsible for local ATMI instance start, stop, configuration test/load/reload, status printing, etc.
tmsrv	Transaction Manager Server. Responsible for XA two phase transaction management.
tprecover	Enduro/X main daemon (ndrxd) monitoring and recovery server.
tmqueue	Persistent queue support server

Table 3.2: Distribution package description - "lib" directory

Resource	Description
libatmi.so	ATMI common client and server shared library, serves functions like tpcall(), tpforward(), etc.
libatmiclt.so	ATMI Client process shared library
libatmisrv.so	ATMI Server process shared library. This contains main() entry point. This serves functions like tpreturn(), tpforward(), etc.
libexnet.so	External network library. Used by tpbridge. This basically is wrapper for sockets. Library provides facility for network clients and network servers. Library allows to handle multiple connections over the single thread
libnstd.so	EnduroX standard library, contains debugging and other facilies
libubf.so	EnduroX Universal Buffer Format handling shared library
libnetproto.so	EnduroX Standard Network protocol library, used by tpbridge

Table 3.3: Distribution package description - "include" directory

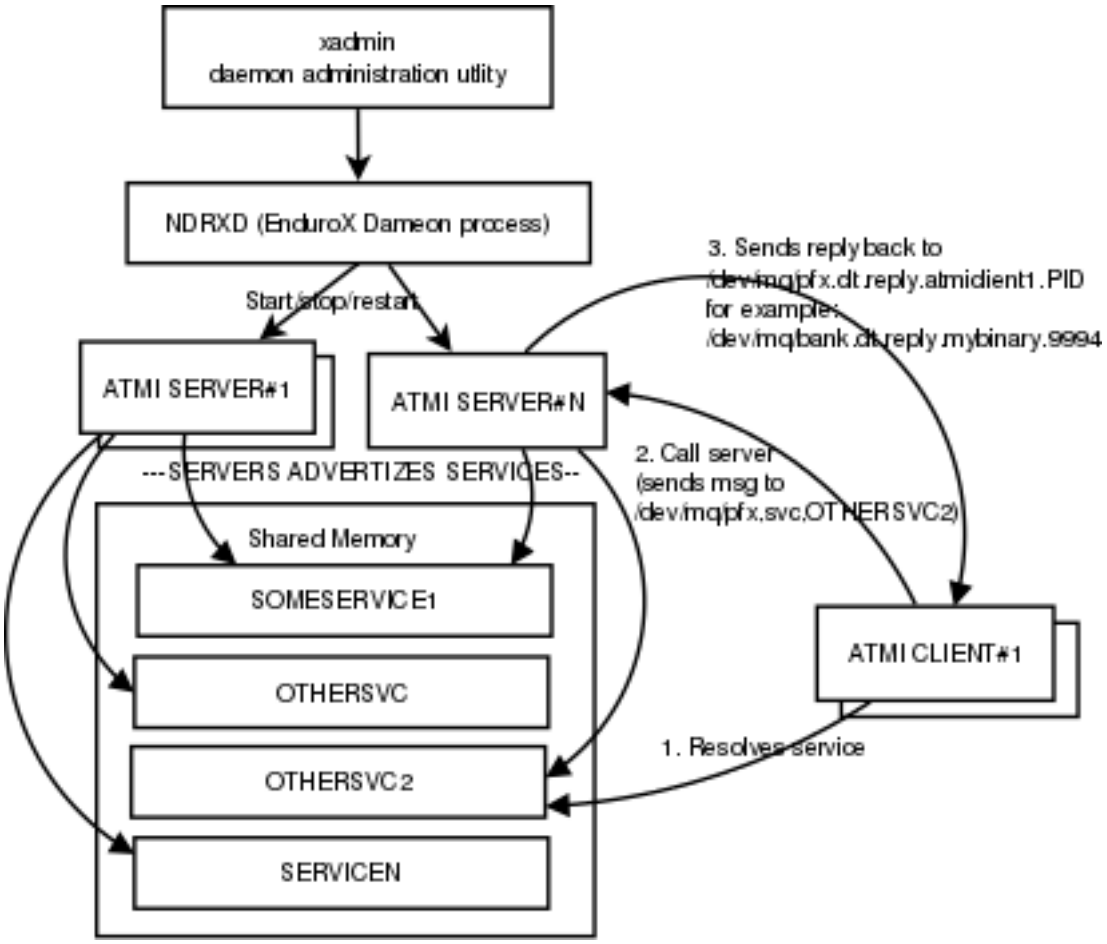
Resource	Description
atmi.h	General ATMI interface
exenv.h	Execution Environment related macros
fml.h	Wrapper for ubf.h
fml32.h	Wrapper for ubf.h
ntimer.h	Timer library
ubf.h	Universal Buffer Format handling header
userlog.h	User logging facility interface

Chapter 4

How system works

Basically local ATMI works by using system's IPC facilities. Following facilities are used by EnduroX:

- System V IPC Semaphores
- Posix Queues
- Posix Sharded Memory



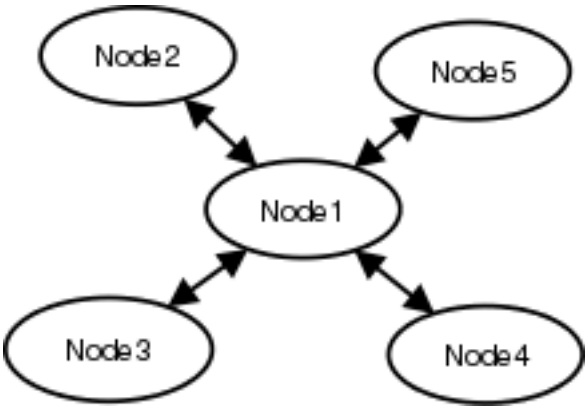
Chapter 5

EnduroX In cluster

This section gives overview how EnduroX Operates in cluster environment. Currently there can be possible 32 nodes in cluster. EnduroX clustering utilises TCP/IP connections to join local EnduroX instances. For each link between two different instances seperate TCP/IP channel is used.

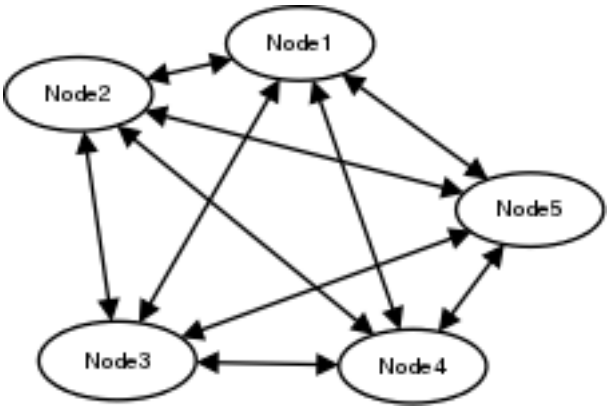
Cluster can be configured in different way, for example with one central node which will have links to all other nodes. Or with no central node, then there should be created links for each of the machine pair.

Cluster with central node:



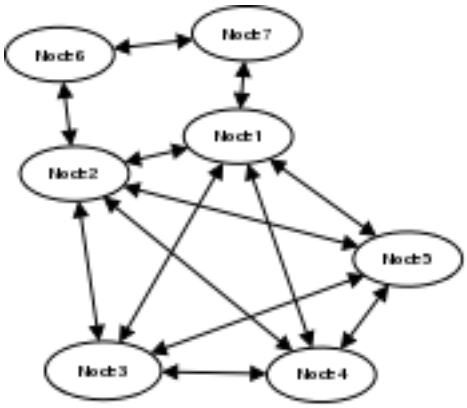
Note that in case of central node, each node only sees centre node (Node1), However centre node sees all other nodes.

Cluster can consist with/out central node, for example this 5 node cluster could look like:



In this case each node sees other each other node and it can create invocations of the services from that node.

Cluster also can consist of mixed node. i.e. when some nodes sees each other but some nodes sees only one or few other nodes. For example consider this 7 node cluster:



In this case Node6 and Node7 sees only few other cluster nodes. Also in this case only Node1 will see Node7 and Node2 will see Node6.

Service tables are replaced only over the direct link. They are not distributed over the whole cluster.

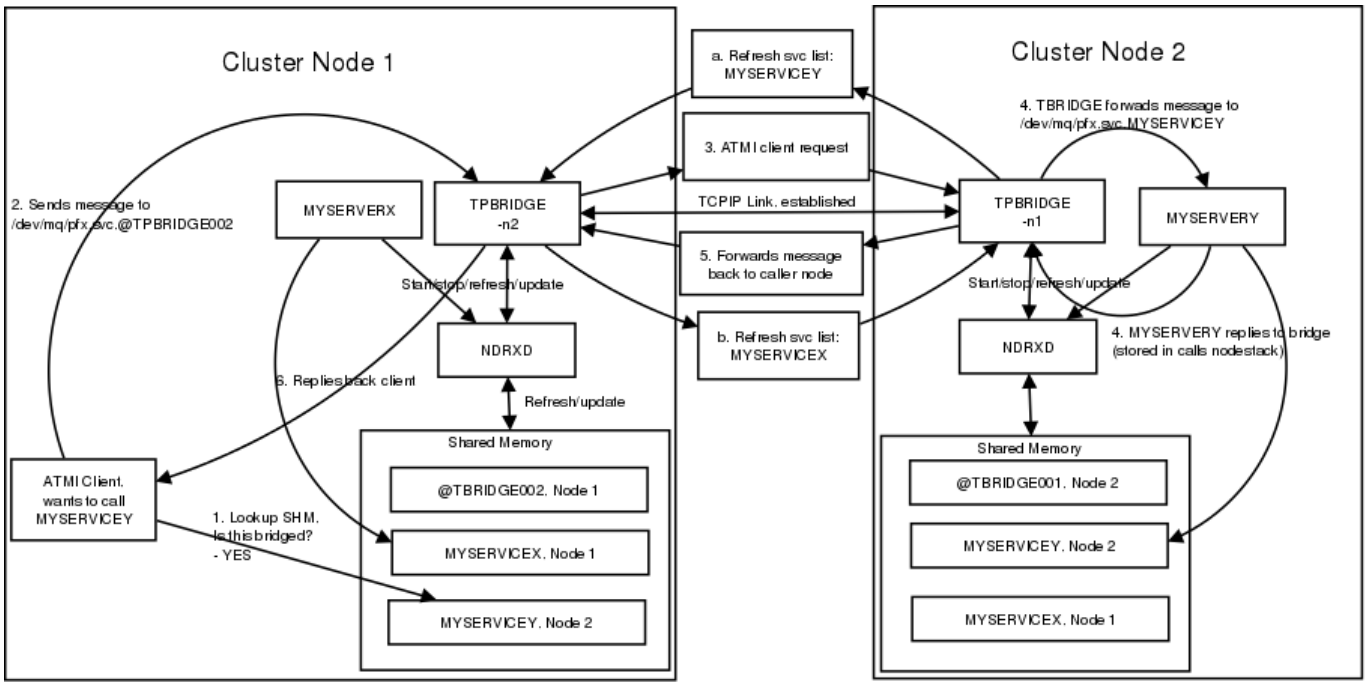
Local EnduroX instances can be cluster by using special ATMI server *tpbridge*. This server accepts configuration (in <*appopts*>) where it says either this endpoint is passive (waits for connection) or active (tries to connect to specified ip address:port). The Node ID of other endpoint and some other parameters.

When connection is established, both endpoints exchanges will full service listings. When some service is remove from local instance, then over this tcp/ip link update message is sent to other node so that service is removed there too.

Full service lists are exchanged periodically (every 30 sec for example). Also *tpbridge* periodically sends zero length messages to other node to keep the connection open.

If connection is lost, both EnduroX local instances will remove all other instance (who's link is lost) services from shared memory.

Here is complete scheme how two nodes cooperate:



As you see firstly when TCP connection is established, service lists are exchanged in points a. and b. (also nodes exchange clock diff so that each call duratation can be corrected between nodes). Each *ndrxd* instance updates shared memory of services received from bridge services.

After that we have ATMI client on Node1 which calls service *MYSERVICEY* which is located on Node2. It resolve shared memory which says that this is on other node. Then call is made to *TPBRIDGE002* Queue, which forwards the packet to other node. See points 1. - 6.

Also it is possible to have service be presented locally and on remote machine. All this information is recorded in shared memory for each of the services. Each shared memory entry contains the 32 element long array which at each cell contains the number of services shared on other node. EnduroX environment parameter *NDRX_LDBAL* says in percentage how much requests serve locally and how much to send to remote machine. Percentage is calculated on random basis and remote node is also calculated on random basis. The shared mem info can be looked by *xadmin*, *psvc* command, for example:

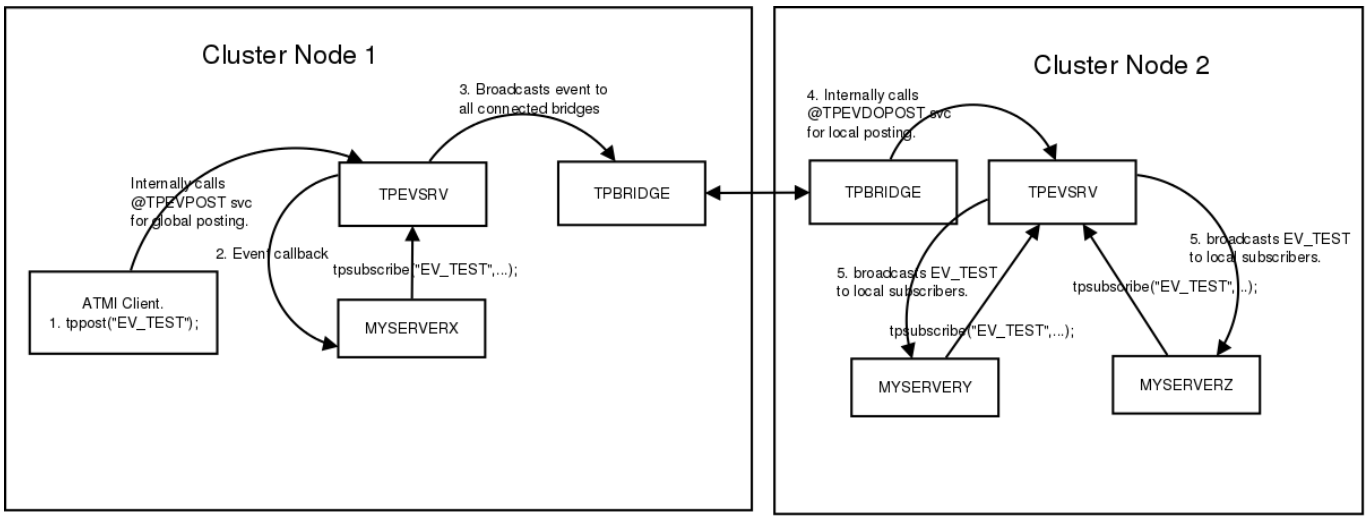
```
$ xadmin
NDRX> psvc
ndrxd PID (from PID file): 5505
Slot   Service Name Nsrv Flags CSrvs TClst CMAX CNODES
------ ------------ ---- ----- ----- ----- ---- --------------------------------
318    RETSOMEDATA  1    1     1     3     12   00000000000300000000000000000000
1051   UNIX2        1    1     1     2     12   00000000000200000000000000000000
3844   @TPEVUNSUBS  1    1     0     0     0    00000000000000000000000000000000
4629   UNIXINFO     1    1     1     3     12   00000000000300000000000000000000
8676   ECHO         1    1     1     3     12   00000000000300000000000000000000
10293  TIMEOUTSV    1    1     1     3     12   00000000000300000000000000000000
11169  @TPEVSUBS    1    1     0     0     0    00000000000000000000000000000000
14301  @TPEVDOPOST  1    1     0     0     0    00000000000000000000000000000000
14894  TESTSV       1    1     1     3     12   00000000000300000000000000000000
16648  @TPBRIDGE002 1    1     0     0     0    00000000000000000000000000000000
16681  @TPBRIDGE012 1    1     0     0     0    00000000000000000000000000000000
17001  NULLSV       1    1     1     3     12   00000000000300000000000000000000
17386  @TPEVPOST    1    1     0     0     0    00000000000000000000000000000000
NDRX>
```

Which for example displays that 2 service instances of *UNIX2* is available on Node12.

Chapter 6

Event processing

EnduroX Supports ATMI events via *tpsubscribe()*, *tppost()* and *tpunsubscribe()* calls. Events are processed by special ATMI server named *tpevsrv*. This server ships in EnduroX package. Events are supported in clustered environment too. In this case the local node additionally broadcasts event to all other connected nodes. On other nodes *tpbridge* process delivers this event to local *tpevsrv* which posts the event locally only.



Chapter 7

Features of EnduroX

This section lists the features of EnduroX framework:

1. Runs on 64bit GNU/Linux, starting from Kernel version 2.6.12.
2. Distributed architecture.
3. Peer based cluster. None of cluster nodes are master.
4. PING of ATMI servers are supported. If server does not respond on pings withing timeframe, they being restarted.
5. EnduroX monitors processes:
    a. For long startup, processes are being killed and restarted
    b. If proceses for some reason dies, they are being restarted
    c. If process dies who was the only which provides some service then SRVCERR response is sent back to caller
    d. For long shutdown (not in time frame), processes are forcebly killed
6. The run-time is possible with out local central ATMI Monitor (ndrxd). As long as other servers are running, system will work.
7. It is possible to restart ndrxd during the runtime. Runtime will not be interrupted. When doing restarting, ndrxd must be started in recovery mode. In this mode it learns about the system and only after a while it becomes a master of the system.
8. Local housekeeping is made. If ATMI clients are unclean shutdown (i.e. not called tpterm()). Then EnduroX daemon detects these cases and cleans up system accordingly.
9. It is easy to debug application for EnduroX. The server processes is possible to start from command line (not mandatory started by ndrxd). This means that it is possible to start server processes via wrappers like valgrind or start via IDE and use step by step debbuging of server process.
10. System is tested by extensive automated unit tests.
11. *buildserver* and *buildclient* are just a wrapper scripts. It is possible to link binaries direclty with correct shared libraries.
12. It is possible to specify environment overrides for each of the seperate ATMI server.
13. EnduroX contains debbuging facilities. It is possible to get debug logs for EnduroX ATMI and UBF sub-systems. Logging for each of the systems can be configured seperately for each of the executables using these libs.
14. ATMI configuration can be reloaded during runtime. It can be done as simple as just editing the config file *ndrxconfig.xml* and running *xadmin reload*.

Chapter 8

Additional documentation

8.1 Internet resources

[1] [ATMI-API] http://docs.oracle.com/cd/E13203_01/tuxedo/tux71/html/pgint6.htm

[2] [FML-API] http://docs.oracle.com/cd/E13203_01/tuxedo/tux91/fml/index.htm

Chapter 9

Glossary

This section lists

ATMI
: Application Transaction Monitor Interface

UBF
: Unified Buffer Format it is similar API as Tuxedo's FML