Overview

Raft.Net is an implementation of Raft consensus algorithm and data exchange between TCP connected nodes for .NET / dotnet.

Raft.Net goal is to make a fault-tolerant platform by making all peers to be in one data state.

Minimal quantity of peers should be 3. Peers are connected between each other via tcp/ip (fully connected network).

All depends upon needs, Raft.Net provides very fast exchange of data between peers in transactional manner, but may be you need DHT/Blockchain or Hashgraph?

Committed data on participating peers of the cluster is always the same and is supplied in the same sequence.
If you intend to receive the same state on all peers ASAP then you need to have as less peers as possible,
Otherwise they will gossip with each other and gossip...
Raft.Net is designed to work efficiently with kind of determined quantity of peers (within the confines of fully connected network)
and designed to bring in one state many entities via the same network transport.

In case of 3-peer-system, when one peer goes down, - system stays functional. For the 5-peer-system, 2 peers can go down and cluster will stay functional.
When majority of peers is functional - the cluster is functional.

System must know in advance total quantity of peers in the cluster to be able to calculate their majority, hence this information must be supplied via configuration.

Quick start

by grabbing and referencing Raft.Net from Nuget.

Implementing IWarningLog interface

```
using Raft;

 static IWarningLog log = null;

        public class Logger : IWarningLog
        {
            public void Log(WarningLogEntry logEntry)
            {
                Console.WriteLine(logEntry.ToString());
            }
        }
```

```
log = new Logger();
```

Create 3 raft nodes made from one configuration (example of configuration), supplying different DBreeze paths (database path for each node) and Different tcp ports (e.g. 4250,4251,4252). First parameter is version of configuration file protocol:

```
TcpRaftNode rn1 = TcpRaftNode.GetFromConfig(1, System.IO.File.ReadAllText("pathToConfig"),
                "pathToDBreezeFolder e.g. D:\Temp\DBreeze\Node1", 4250, log,
                (entityName, index, data) => { Console.WriteLine($"Committed {entityName}/{index}"); return true; });

TcpRaftNode rn2 = TcpRaftNode.GetFromConfig(1, System.IO.File.ReadAllText("pathToConfig"),
                "pathToDBreezeFolder e.g. D:\Temp\DBreeze\Node2", 4251, log,
                (entityName, index, data) => { Console.WriteLine($"Committed {entityName}/{index}"); return true; });

TcpRaftNode rn3 = TcpRaftNode.GetFromConfig(1, System.IO.File.ReadAllText("pathToConfig"),
                "pathToDBreezeFolder e.g. D:\Temp\DBreeze\Node3", 4252, log,
                (entityName, index, data) => { Console.WriteLine($"Committed {entityName}/{index}"); return true; });

rn1.Start();
rn2.Start();
rn3.Start();
```

To put a command to the cluster use:

```
rn1.AddLogEntry(new byte[] { 23 })
Or
rn2.AddLogEntry(new byte[] { 27 })
Or
rn3.AddLogEntry(new byte[] { 29 })
```

When all nodes come to the consensus about the command, this command will be thrown back into the OnCommit function (we have described it here like this)

```
(entityName, index, data) => { Console.WriteLine($"Committed {entityName}/{index}"); return true; }
```

Raft.Net will throw one by one committed commands.

OnCommit function is called on all nodes and on the **same sequence**, so after getting it you can **implement** your **business logic** connected to this command (e.g command can contain set of insert/remove parameters or an intent to increase kind of a value by 1).

When OnCommit function ends up by “return true;”, the next committed command will be supplied to OnCommit function.
If node restarts before returning “true” inside of OnCommited function (or just wakes up after long sleep), it will start to receive OnCommited from the same Last committed place to get a chance to finalize its business logic cycle.

Size of the command is internally limited to 5MB, but that can be easily changed.

Call of **`AddLogEntry` doesn’t give a guarantee that command will be executed** - something can happen with the spreading leader or in the middle of the sending tract. That’s why if OnCommit doesn’t return this command back within kind of time interval - correspondent actions should be taken.
Call of **OnCommit means a strong consistency guarantee** - you can be sure that command has reached majority of peers (afterwards all other peers will be covered) and business logic actions will be executed in the same sequence. This gives the same entity state to all configured peers.

If `AddLogEntry is` returned by timeout and your client has generated another `AddLogEntry request` (in fact, it is another command with another id, but with the same content as timed out one), **your client software should handle the case if both commands will be returned correctly (business logic part), despite the timeout of one of them**.

Entities

When we talk about consensus we talk about the consensus of the state of one entity.
Raft.Net supports multiple entities to be synchronized via one Tcp transport. It means that Raft.Net peers of one cluster can have multiple synchronized entities.
Additional entities are configured via [configuration file](#).
They are accessible via EntityName:

```
rn1.AddLogEntry(new byte[] { 23 }, entityName: "inMemory1")
```

Persistence

Currently supported:

- standard persistence on disk
- persistence on disk with flush interval (can be good in case if the command stream is very high and backend is HDD)
- Persistence in memory
- Persistence in memory when only latest entity state is loaded after node restarting (it will not load the whole history, but only latest state - acts like high speed cache)

Configuration

```
//GlobalConfig

EndPoint: 127.0.0.1,4250
EndPoint: 127.0.0.1,4251
EndPoint: 127.0.0.1,4252
EndPoint: 127.0.0.1,4253
EndPoint: 127.0.0.1,4254

//EOF GlobalConfig

//Starting configs per entity
//First entity should be always with the name "default" or it will be auto-renamed to "default""
//Raft.RaftNodeSettings

Entity: default
VerboseRaft: false
VerboseTransport: false
DelayedPersistenceIsActive: true
DelayedPersistenceMs: 10000
InMemoryEntity: false
InMemoryEntityStartSyncFromLatestEntity: false

Entity: inMemory1
VerboseRaft: true
VerboseTransport: true
DelayedPersistenceIsActive: false
InMemoryEntity: true
InMemoryEntityStartSyncFromLatestEntity: false
```

List of all endpoints of the Raft.Net cluster

Debug message for IWarningLog

Increases speed of processing when storage is HDD.

When DelayedPersistenceIsActive set to true, sets up flush on the disk interval, should be set from 3-10 seconds.

When node restarts this parameter set to true, helps to load only latest committed state for the entity, not the complete history. Works only in-memory mode

Examples

Raft\Raft_NodeTest\bin\Debug contains configurations for 5 nodes. Path to DBreeze can be changed.
Started all these files

_NodeTest.exe	28.02.2018 16:39	Приложение	8 КБ
_NodeTest.exe.config	05.02.2018 11:30	XML Configuratio...	1 КБ
_NodeTest.pdb	28.02.2018 16:39	Program Debug D...	16 КБ
DBreeze.dll	20.02.2018 12:06	Расширение при...	370 КБ
DBreeze.xml	20.02.2018 12:06	Файл "XML"	336 КБ
Raft.dll	28.02.2018 16:39	Расширение при...	83 КБ
Raft.pdb	28.02.2018 16:39	Program Debug D...	212 КБ
raftConfig.txt	28.02.2018 17:35	Файл "TXT"	1 КБ
start4250.bat	28.02.2018 13:39	Пакетный файл ...	1 КБ
start4251.bat	28.02.2018 13:39	Пакетный файл ...	1 КБ
start4252.bat	28.02.2018 11:21	Пакетный файл ...	1 КБ
start4253.bat	28.02.2018 11:21	Пакетный файл ...	1 КБ
start4254.bat	28.02.2018 11:21	Пакетный файл ...	1 КБ

```
C:\WINDOWS\system32\cmd.exe
0\Projects\blaze\Raft\Raft\_NodeTest\bin\Debug\raftConfig.txt; Path to DBreeze f
older: D:\Temp\RaftDBreeze\Node4251
51:49.5149> Node 4251 has started.
28.02.2018 17:51:49> [DEBUG] [] [] [Started TcpNode on port 0.0.0.0:4251]
51:54.5154> Node 4251 LeaderHeartbeatTimeout
51:54.5154> Node 4251 RunElectionTimer 372 ms
51:54.5154> Node 4251 election timeout
51:54.5154> Node 4251 state is Candidate _ElectionTimeout
51:54.5154> Node 4251 RunElectionTimer 394 ms
51:55.5155> Node 4251 election timeout
51:55.5155> Node 4251 state is Candidate _ElectionTimeout
51:55.5155> Node 4251 RunElectionTimer 437 ms
51:55.5155> Node 4251 state is Leader _ParseVoteOfCandidate
51:55.5155> Node 4251 is Leader ********************************************
set1
Adding: LOG_ENTRY_IS_CACHED
wow committed default/34
set1
Adding: LOG_ENTRY_IS_CACHED
wow committed default/35
set1
Adding: LOG_ENTRY_IS_CACHED
wow committed default/36
wow committed default/37
```

```
C:\WINDOWS\system32\cmd.exe
C:\Users\blaze\Documents\Visual Studio 2010\Projects\blaze\Raft\Raft\_NodeTest\b
in\Debug>"C:\Users\blaze\Documents\Visual Studio 2010\Projects\blaze\Raft\Raft\_
NodeTest\bin\Debug\_NodeTest.exe" 1 4250 "C:\Users\blaze\Documents\Visual Studio
 2010\Projects\blaze\Raft\Raft\_NodeTest\bin\Debug\raftConfig.txt" "D:\Temp\Raft
DBreeze\Node4250"
Scenario 1 is running
Listening port: 4250; Path to config: C:\Users\blaze\Documents\Visual Studio 201
0\Projects\blaze\Raft\Raft\_NodeTest\bin\Debug\raftConfig.txt; Path to DBreeze f
older: D:\Temp\RaftDBreeze\Node4250
52:29.5229> Node 4250 has started.
28.02.2018 17:52:29> [DEBUG] [] [] [Started TcpNode on port 0.0.0.0:4250]
wow committed default/34
wow committed default/35
wow committed default/36
set1
Adding: NODE_NOT_A_LEADER
wow committed default/37
```

```
C:\WINDOWS\system32\cmd.exe
C:\Users\blaze\Documents\Visual Studio 2010\Projects\blaze\Raft\Raft\_NodeTest\b
in\Debug>"C:\Users\blaze\Documents\Visual Studio 2010\Projects\blaze\Raft\Raft\_
NodeTest\bin\Debug\_NodeTest.exe" 1 4254 "C:\Users\blaze\Documents\Visual Studio
 2010\Projects\blaze\Raft\Raft\_NodeTest\bin\Debug\raftConfig.txt" "D:\Temp\Raft
DBreeze\Node4254"
Scenario 1 is running
Listening port: 4254; Path to config: C:\Users\blaze\Documents\Visual Studio 201
0\Projects\blaze\Raft\Raft\_NodeTest\bin\Debug\raftConfig.txt; Path to DBreeze f
older: D:\Temp\RaftDBreeze\Node4254
51:52.5152> Node 4254 has started.
28.02.2018 17:51:52> [DEBUG] [] [] [Started TcpNode on port 0.0.0.0:4254]
51:53.5153> Node 4254 RunElectionTimer 288 ms
51:53.5153> Node 4254 (Follower) VoteFor 4250  in  _ParseCandidateRequest
51:54.5154> Node 4254 (Follower) VoteReject 4251  in  _ParseCandidateRequest
51:55.5155> Node 4254 RunElectionTimer 241 ms
51:55.5155> Node 4254 (Follower) VoteFor 4251  in  _ParseCandidateRequest
wow committed default/34
wow committed default/35
wow committed default/36
wow committed default/37
```

```
C:\WINDOWS\system32\cmd.exe
C:\Users\blaze\Documents\Visual Studio 2010\Projects\blaze\Raft\Raft\_NodeTest\b
in\Debug>"C:\Users\blaze\Documents\Visual Studio 2010\Projects\blaze\Raft\Raft\_
NodeTest\bin\Debug\_NodeTest.exe" 1 4252 "C:\Users\blaze\Documents\Visual Studio
 2010\Projects\blaze\Raft\Raft\_NodeTest\bin\Debug\raftConfig.txt" "D:\Temp\Raft
DBreeze\Node4252"
Scenario 1 is running
Listening port: 4252; Path to config: C:\Users\blaze\Documents\Visual Studio 201
0\Projects\blaze\Raft\Raft\_NodeTest\bin\Debug\raftConfig.txt; Path to DBreeze f
older: D:\Temp\RaftDBreeze\Node4252
51:50.5150> Node 4252 has started.
28.02.2018 17:51:50> [DEBUG] [] [] [Started TcpNode on port 0.0.0.0:4252]
51:53.5153> Node 4252 RunElectionTimer 205 ms
51:53.5153> Node 4252 (Follower) VoteFor 4250  in  _ParseCandidateRequest
51:54.5154> Node 4252 (Follower) VoteReject 4251  in  _ParseCandidateRequest
51:55.5155> Node 4252 RunElectionTimer 222 ms
51:55.5155> Node 4252 (Follower) VoteFor 4251  in  _ParseCandidateRequest
wow committed default/34
wow committed default/35
wow committed default/36
wow committed default/37
```

```
C:\WINDOWS\system32\cmd.exe
C:\Users\blaze\Documents\Visual Studio 2010\Projects\blaze\Raft\Raft\_NodeTest\b
in\Debug>"C:\Users\blaze\Documents\Visual Studio 2010\Projects\blaze\Raft\Raft\_
NodeTest\bin\Debug\_NodeTest.exe" 1 4253 "C:\Users\blaze\Documents\Visual Studio
 2010\Projects\blaze\Raft\Raft\_NodeTest\bin\Debug\raftConfig.txt" "D:\Temp\Raft
DBreeze\Node4253"
Scenario 1 is running
Listening port: 4253; Path to config: C:\Users\blaze\Documents\Visual Studio 201
0\Projects\blaze\Raft\Raft\_NodeTest\bin\Debug\raftConfig.txt; Path to DBreeze f
older: D:\Temp\RaftDBreeze\Node4253
51:51.5151> Node 4253 has started.
28.02.2018 17:51:51> [DEBUG] [] [] [Started TcpNode on port 0.0.0.0:4253]
51:53.5153> Node 4253 RunElectionTimer 585 ms
51:53.5153> Node 4253 (Follower) VoteFor 4250  in  _ParseCandidateRequest
51:54.5154> Node 4253 (Follower) VoteReject 4251  in  _ParseCandidateRequest
51:55.5155> Node 4253 RunElectionTimer 514 ms
51:55.5155> Node 4253 (Follower) VoteFor 4251  in  _ParseCandidateRequest
wow committed default/34
wow committed default/35
wow committed default/36
wow committed default/37
```

And there several commands described in [Program.cs]

```
AddLogEntryResult addRes = null;
while(true)
{
    var cmd = Console.ReadLine();
    switch(cmd)
    {
        case "set1":
            addRes = rn.AddLogEntry(new byte[] { 23 });
            Console.WriteLine($"Adding: {addRes.AddResult.ToString()}");
            break;
        case "set1a":
            addRes = rn.AddLogEntry(new byte[] { 27 },entityName: "inMemory1");
            Console.WriteLine($"Adding: {addRes.AddResult.ToString()}");
            break;
        case "set10":
            for (int k = 0; k < 10; k++)
            {
                addRes = rn.AddLogEntry(new byte[] { 23 });
                Console.WriteLine($"Adding: {addRes.AddResult.ToString()}");
            }
            break;
        case "set10a":
            for (int k = 0; k < 10; k++)
            {
                addRes = rn.AddLogEntry(new byte[] { 23 }, entityName: "inMemory1");
                Console.WriteLine($"Adding: {addRes.AddResult.ToString()}");
            }
            break;
    }
}
```

Satellite assemblies

DBreeze and integrated Biser.Net are used to make Raft.Net possible and can be reused in your project also.