This document summarizes Distributed Transaction Manager discussions in the Mero architecture team. The goal is to provide a starting point for design level discussions and resource estimation. Note that this documents is not a substitute for a list of requirements or use-cases. Intended audience includes T1 team and Mero architects.

Overview.

Distributed Transaction Manager (DTM) is a component supporting distributed transactions, which are groups of file system updates that are guaranteed to be atomic w.r.t. failures. This document only describes low level transaction mechanism. Higher level mechanisms such as storage application resumption and state synchronization after a failure, and non-blocking availability are described elsewhere. See DTM View Packet and Distributed transactions QAS list for more information.

Items.

- Mero (as well as other cluster file systems) scatters dependent data and meta-data across multiple nodes for fault-tolerance (redundancy) and scalability (striping) purposes. Examples of placing dependent state on different nodes are:
    - clustered meta-data, *e.g.*, a file inode located on a server different from a server where parent directory lives;
    - striping, where file data are located on servers different from ones where meta-data inode is;
    - mirroring (or, generally speaking, redundant striping).

  In these and other cases maintaining system state consistency requires that a state on multiple nodes is updated atomically. Atomicity means two (related) things:

    - concurrency control: other users see either all related updates or none of them;
    - failures: either all related updates or none of them survive a failure.

  It's not a coincidence that these issues of concurrency control and failure atomicity are interrelated. We shall focus on failure atomicity in this discussion.

- The group of updates that must survive a failure atomically is called *a distributed transaction*. The term transaction is widely used in the data-base world. Data-base transactions, are called *ACID* transactions, because they possess the following properties: atomicity (A, this is what we are talking about), consistency (C, a transaction preserves system consistency. Our transactions have this property too, but DTM has nothing to do with it), isolation (I, meaning that transactions are isolated from effects of each other. Again, Mero distributed transactions can have this property, but it is beyond DTM to guarantee this.), and durability (D, meaning that once a transaction has completed successfully, it will survive any further failure). Of all these properties, Mero distributed transactions have only atomicity.

- To understand why file system transactions are not usually durable, one has to keep in mind that a typical data-base transaction is an SQL statement, which can involve table scans, distributed sorting and other complicated, expensive and lengthy actions. While the cost of making a transaction durable when it completes is justified for such large transactions, a typical

file system transaction is much smaller: it's typically a very short action, like inserting a name into a directory and initializing an inode in an inode table. Making short transactions durable on completion would add an intolerable overhead to file system activity.

- Mero DTM provides an interface to group a collection of file system updates into a distributed transaction. The updates of transaction are then sent over the network as RPCs and executed by their respective servers. DTM adds certain information to each update that allows servers to identify which updates are part of the same transaction and to act appropriately.

- DTM guarantees that should a server node fail and restart, the effects of distributed transactions that have updates on this server are either completely restored after restart or completely eliminated.

- To achieve this atomicity property, DTM uses a well-known technique of Write-Ahead Logging (WAL): before executing an operation, a server puts into its FOL a record, describing this operation and distributed transaction the operation is a part of. On a restart, the server goes through the FOL and undoes or redoes the operations from it to achieve atomicity.

- Clearly, the restart process described above needs some mechanism by which servers can agree which operations are to be restored and which are to be eliminated. Mero has two such mechanisms: redo-only recovery and redo-undo recovery.

- Please see this presentation for details.