



# Revealing the Magic Behind NonStop Systems

( or  
The 7 Fundamentals As Applied To  
TMF + DP2 + NSK + RDF + ISC )

Charles Johnson  
TMF Architect  
NonStop Software

July 22, 2002

# 7 Fundamentals

Data integrity (data must be checked wherever it goes)

Reliability (FD + FT + FA)

Parallelism (if it isn't locked, it isn't blocked)

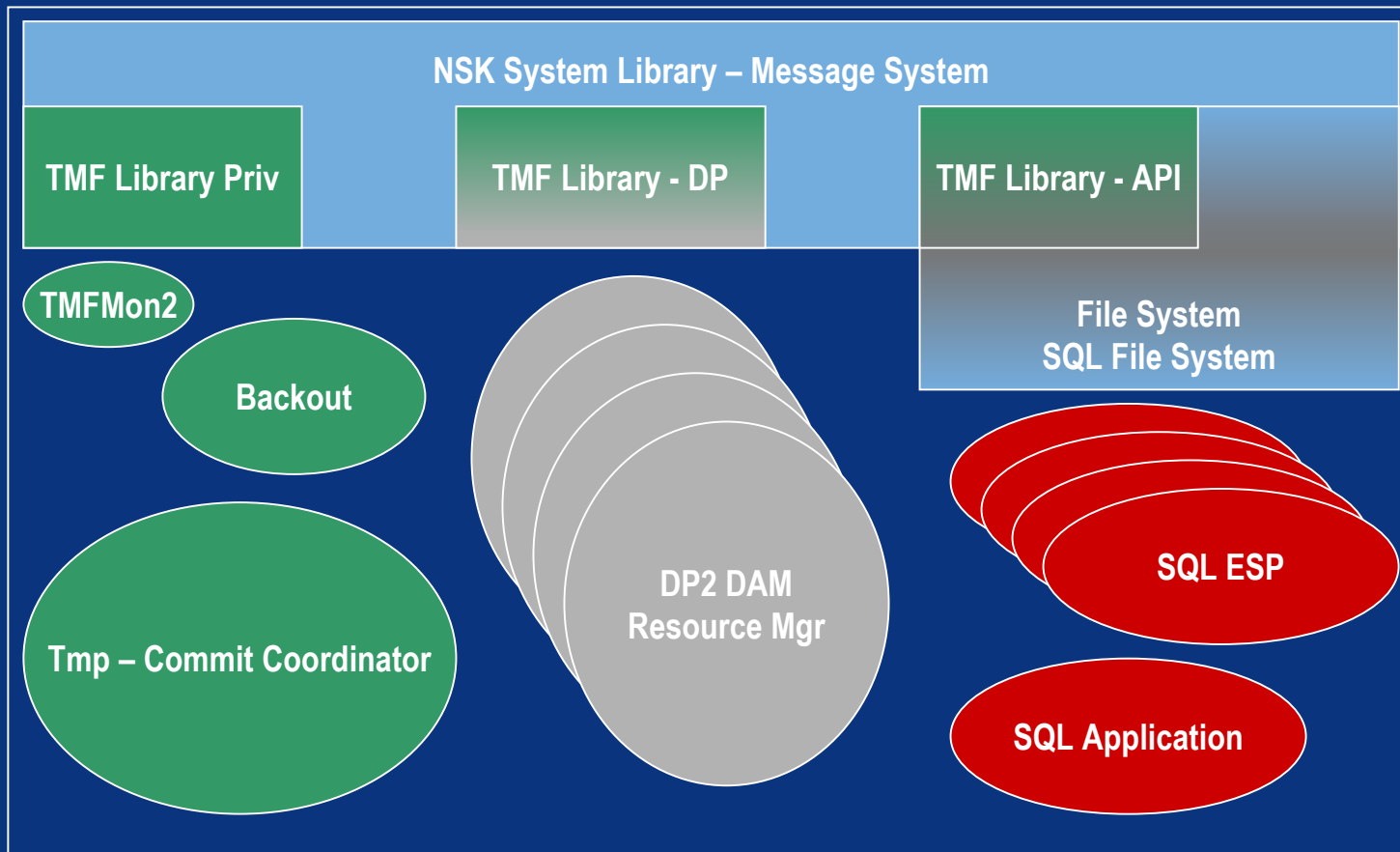
Transparency (when/where/how)

Scalability (cluster and network)

Availability (outage minutes -> zero)

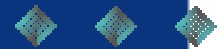
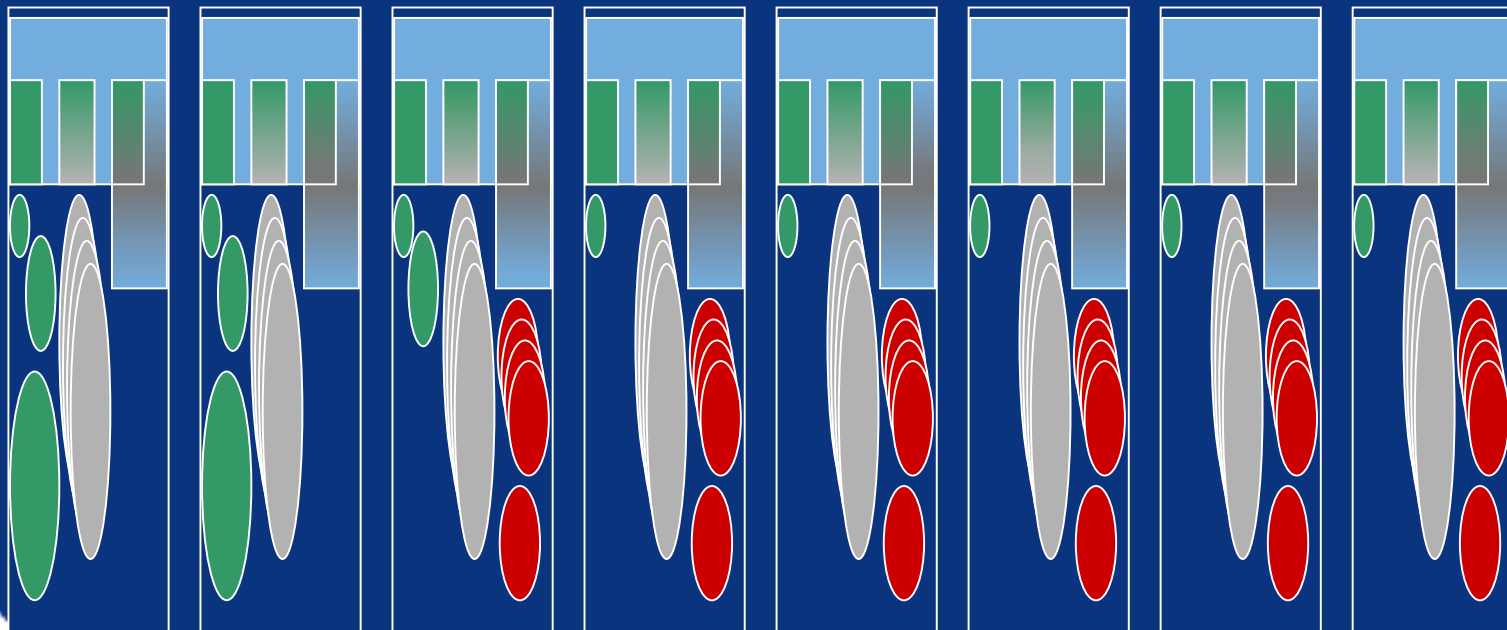
Database consistency (database must be serialized wherever it goes)

# A Single NSK CPU Can Contain ...



# An NSK Node Can Look Like ...

*ServerNet*



# Data Integrity (Data Must Be Checked Wherever It Goes)

Data corruption is an ever-present possibility through electronic noise (e.g. radon decay chain effects, cosmic radiation), physical defects (semiconductor doping flaws), and HW/SW design defects (stray pointers in code)

The statistics are that there are 3 undetected and uncorrected, but program-significant data corruptions per 1000 microprocessors per year (Horst, et al Proc 23rd FT Computing Symposium 1993)

# Data Integrity (Data Must Be Checked Wherever It Goes) ...

NSK memory is error checking and correcting (ECM), like most computer systems. Many other components have the potential to corrupt data, and this will get worse as components shrink

Nonstop systems are unique in supporting every one of the following:

- Lock-stepped microprocessors
- Complete protection of internal buses and drivers
- end-to-end checksums on data sent to storage devices

# Data Integrity (Data Must Be Checked Wherever It Goes) ...

Log writing in the ADP uses end-to-end checksums on blocks (in an upcoming release). This is because after a crash, we need to fix up to the last valid written block of audit from an audit buffer, and we can't tolerate garbage in the block middle due to power-loss partial writes

During TMF restart after a TMF crash or full system crash, "fixup" then searches for the last good block written (valid checksum), which becomes the new log tail

# Reliability (FD + FT + FA)

James Gosling: distributed computing is not transparent to either failure or performance

Some errors are tolerable and some operations returning errors can be retried with idempotence

“Fail fast” – sometimes going down quickly prevents the spread of invalid data or even the effects of flawed algorithms or races we can't handle (what if the corruption checks don't catch something?)

# Reliability (FD + FT + FA) ...

FD = failure detection: assertion logic is interwoven throughout all critical NonStop code

To maintain the state machine invariants end-to-end we must detect violation of the invariants

FT = fault tolerance: when something goes wrong and a failure occurs, whether hardware or software, takeover mechanisms ensure the re-establishment of state machine invariants (a new state which is equivalently correct to the state before failure)

# Reliability (FD + FT + FA) ...

In fault tolerant systems, when a batch of hardware is faulty, the fault tolerance of the software has to function correctly

FA = fault avoidance: by small amounts of forethought and action here and there in the code, potentially large failures can be shrunk down in size to be handled invisibly (e.g. avoiding unnecessary aborts by shipping tx/dp crosslinks to the backup dp cpu, or avoiding unnecessary volume recovery outages by detecting lost audit on a transaction basis, etc.)

# Parallelism (If It Isn't Locked, It Isn't Blocked)

NS SQL MP and MX both use strict 2PL (2 phase locking) for the transaction duration locks

The disk process (SQL resource manager) holds both the data and the locks, there is no external distributed lock management

So, one message connects the transactional SQL code with the data + the acquired lock + the tx/dp crosslink within the node transaction service (TMF) underneath the disk process + the backup disk process (via the interrupt service level of the system)

# Parallelism (If It Isn't Locked, It Isn't Blocked) ...

There is one copy of the data in the NS SQL universe: applications must serialize, they must interact with each other on the same data using transactions: this is in contrast to versioning databases (Oracle, MS Sql Server, Sybase, Illustra/Informix) where transactional reads are from version cache and are blind to concurrent updates: so only updates block updates

Update locks block updates and reads, shared read locks block updates, and both are released only when changes to the database are made durable at commit or abort time

# Parallelism (If It Isn't Locked, It Isn't Blocked) ...

This allows every process to run freely parallel until they encounter a blocking lock on some record – so there is no application locking schedule or high level concurrency control required – so the system runs completely in parallel at warp speed!!!!

In TMF, the read-only optimization further allows large sections of the database to be released at the beginning of the commit flush (which is the end of the database transformation from data that was read)

# Transparency (When/where/how)

Gosling: distributed computing is not transparent to either failure or performance (once again !)

Transparent / opaque to whom?

At the TMF system programming level, there is no clustering or failure transparency, precisely to provide that transparency (whenever possible) to the layers above

# Transparency (When/where/how) ...

For the vast majority of hardware and software failures, even most double failures, TMF seamlessly rolls along without even aborting transactions, so the applications don't need to worry about those failures

Occasionally, the seamless operations and functioning of the application above are interrupted, as in a double failure of a data volume, since TMF doesn't know who touched what data volume after both halves of one go down

# Transparency (When/where/how) ...

Then TMF will have to abort transactions to restore the state of the database, and this will require applications to resubmit any uncommitted updates (like the current mini-batch at NASDAQ)

If the process that begins a transaction dies it gets aborted and work must be resubmitted

Finally, if there is a TMF crash, due to a double disk or cpu failure of the log, then TMF must be restarted or the backup node must take over in the case of ISC (proposed) or RDF (neither is very transparent to applications)

# Transparency (When/where/how) ...

So when/where/how is this transparent to the application ?

- In the consistent view of the database outside of a transaction that either commits or aborts
- In the guarantee that if the transaction service says commit and the system takes a nosedive a nanosecond later, then the transaction data is there

# Transparency (When/where/how) ...

So when/where/how is this transparent to the application  
? ...

- In the guarantee that if the transaction service says abort, then all transaction protected work is undone completely before any transaction locks are released
- In that the application needs to do nothing, but use a transaction to guarantee all of that consistency

# Scalability (Cluster and Network)

Scalability of database logging performance inside the node and for node pairs (RDF and the ISC proposal) is accomplished by the three phase flushing algorithm and the forced commit write by the commit coordinator

Data volumes never force-write database updates to the log, instead those updates are streamed to the log (ADP) input buffer

# Scalability (Cluster and Network) ...

Data volumes use the WAL (write ahead log) protocol so that writes only have to be scheduled to the data volume disk every five minutes or so (called DP control points or disk checkpoints) giving “in-memory” update database performance for the data volume disk

But not quite “in-memory” performance, because of the five minute rule: Keep a data item in electronic memory if its access frequency is five minutes or higher; otherwise keep it in magnetic memory. (Gray/Reuter 2.2.1.3) This rule will apply as long as the price ratio between e/m memory stays reasonably constant

# Scalability (Cluster and Network) ...

At commit time, the node's transaction service induces explicit data volume flushing only when necessary, from the interrupt service level of ServerNet (100 times cheaper than message system wakeups). Since, in busy systems the data volumes are stream-writing ahead continuously to the log, then the transaction updates are almost always already flushed to the log when commit time comes (for big transactions)

When flushes are reported to the commit coordinator (Tmp) in a busy system, they are lumped together into a single and periodic forced write into the log (group commit)

# Scalability (Cluster and Network) ...

The group commit write by the Tmp is the one and only time in the system that the transactional database application absolutely must wait for the disk to spin and the drive head to move, and it's a shared experience (and thereby scalable for the node's transaction service)

So, why is writing to a one log disk (ADP) faster than writing to a bunch of data volume disks? If there is no other activity on the disk, and if we write it sequentially using big buffers: then by treating a disk like a tape we get 20-100 times the writing throughput (Gray/Reuter 2.2.1.2).

# Scalability (Cluster and Network) ...

Ultimately, however, you can easily generate more audit than one log disk (audit trail) can take, so we've partitioned the log 16 ways (TMF merged audit). But you still only force write one commit buffer to the Master Audit Trail (MAT) disk while streaming audit to up to 15 auxiliary audit trails.

So, part of the configuration of a node's TMF transaction service is to assign data volumes to auxiliary audit trails. Reassigning data volumes to log to other auxiliary ADPs does not require the transaction service to be brought down, and can be done online

# Scalability (Cluster and Network) ...

When transactions span the network to other NSK nodes (or heterogeneously to other vendor systems), then the commit coordinators on the two nodes do non-blocking two phase commit to guarantee the joint commit or abort of the transaction

After much work on the scalability of network transactions: on S74016 systems connected via ServerNet Clusters (Starburst) and using Turbo TMF/MMI, we can now do 60% of the local maximum transaction rate across as many nodes as the customer needs. That's called "scaling out"

# Scalability (Cluster and Network) ...

If transactions have locality of reference and only touch the local node with no lock conflicts (as in the NASDAQ 1000 TPS server implementation), then NSK systems do “scaling up” at nearly the full 100% level

Based on the scalability of TMF and the log, RDF consistently maintains the database on a second node with only 1% overhead + 4% expand overhead on the primary node (the 4% is expected to drop using ServerNet Clusters: Starburst)

# Scalability (Cluster and Network) ...

RDF consumes more of the backup system applying the updates (between 15% and 25%), but that overhead is also expected to drop substantially in the Independent Product Release 1.2, which is to be deployed using accelerated code

# Availability (Outage Minutes -> Zero)

What is availability?

On some systems it is defined as the existence of a working Unix shell prompt

On some databases (Oracle) it is measured only on database software-produced outages, as though hardware and operating system-produced outages that are not tolerated by the database system are somehow not really happening to the customers

# Availability (Outage Minutes -> Zero) ...

In TMF, availability is measured in terms of database queuing: *if you can begin a transaction, and queue up for a lock on any part of the database under that transaction with the likelihood of getting that lock and accessing that data, then the database is considered available*

If you can't do all that on some part of the database, then that part of the database is considered unavailable

# Availability (Outage Minutes -> Zero) ...

TMF has had excellent fault tolerance for nearly 20 years with non-blocking two phase commit coordination between NonStop server nodes: when the Tmp process or cpu dies, the backup Tmp takes over with no perceptible outage or loss of state or any transactions being aborted

# Availability (Outage Minutes -> Zero) ...

Before the D30 release of NSK, TMF2 would do a TMF crash if more than one cpu went down at the same time (TMF only tolerated single failures). Then the customer had to cold load the system, because we didn't know how to clean up the mess in the cpu global memory resident data structures

Since D30, TMF3 handles multiple cpu failures transparently, as long as one-half of the \$system disk, one-half of any of the audit trail disks and one-half of the \$Tmp process pair stays up

# Availability (Outage Minutes -> Zero) ...

In the D30 TMF3 release many operator induced outages and limitations were removed from TMF:

- You can't delete audit trail files by accident, and AT space is claimed when configured (no more error 43 hangs)
- If correctly configured, TMF won't hang up due to an operator neglecting to hang a tape: by declaring overflow audit trail disks on the fly
- You can increase audit trail space on disk on the fly
- You can add active volumes to a multi-volume audit trail on the fly
- With support for DSC for disks, you can pull data volume disks from one cpu and just insert them into another cpu (physically moving the hot spot disk)

# Availability (Outage Minutes -> Zero) ...

In the D30 TMF3 release, Backout, Volume Recovery and File Recovery performance all increased by a factor of 30, with the elimination of the behavior of writing through to disk: This reduces the outage minutes on parts of the database that are in trouble

There are more of these online 24X7 operational improvements coming in future releases of TMF (the group is doing the more difficult ones, now)

# Availability (Outage Minutes -> Zero) ...

IBM's DB2 uses RIDs to identify records in btrees, we use logical keys: this means our btrees can be moved without modification of the btree data, IBM RIDs are only valid in one container file and need to be remapped to move them. This causes holes in the implementation.

This is why in NS SQL, we can split, merge, and move partitions, and reorganize the database on the fly without any availability outage, using TMF recovery interfaces to pull the updates to the old partition from the log in real time and apply them to the new partition, and then switch over

# Availability (Outage Minutes -> Zero) ...

NS SQL's ability to do online operations like these is unique in the industry, and these features are all transparently fault tolerant and retryable after failure (of course)

But, of course, nothing is perfect: TMF can and will crash, or the NSK system can crash and then your database is unavailable, so what then?

# Availability (Outage Minutes -> Zero) ...

RDF propagates the MAT and up to 15 auxiliary audit trails of changes to files on an RDF backup node, and keeps those RDF-protected files in sync until a system outage, or operational need arises which requires a switchover to the backup node

RDF, in the Independent Product Release 1.2, will be capable of doing takeovers in less than one minute: well within highly available “system crisis” response times that are acceptable in the industry

# Availability (Outage Minutes -> Zero) ...

But, if the disks are 90% of the cost of very large systems, isn't duplicating all the drives on a backup node incurring 190% of the cost of a system? Hence, the ISC (proposed)

The Indestructible Scalable Computing initiative potentially solves this problem by one proposed solution of having one set of disks on a ServerNet SAN, which a primary and backup node can both access

# Availability (Outage Minutes -> Zero) ...

Then according to that design proposal, on the primary node, DP2 pairs and the Tmp pair do network checkpoints to their partner pairs on the backup node, keeping them in complete synchrony

At system failure time, or by coordinated switchover, the backup node can take over the disks, the backup node Tmp can populate the cpu global memory data structures with good-for-any-transaction-during-restart tx/dp crosslinks, and the system can takeover immediately, at lightning speed without aborting transactions

# Database consistency (database must be serialized wherever it goes)

So what is database consistency?

It's measured by pH: the higher the ACIDity, the better

The letters in ACID stand for Atomic, Consistent, Isolated, and Durable

# Database consistency (database must be serialized wherever it goes) ...

**A is for Atomicity** and means all or nothing: the database everywhere must end up in a state whose visibility to the world outside the transaction is first the old state, then the new state (in the case of commit) or the state remains unchanged (in the case of abort)

# Database consistency (database must be serialized wherever it goes) ...

C is for Consistency seems like a circular definition, but actually it should probably be spelled ASID, because consistency in database work is really serialization

The database (at least on a NonStop system) is really in the log, the data volumes are a convenient cache whose disk image is only rarely in a consistent state (only after a correctly completed shutdown of the node transaction service, at which point it's unusable)

Serialization in the log is defined by the exclusive existence of serializable transaction histories, and no other kind

# Database consistency (database must be serialized wherever it goes) ...

A transaction history starts in the log with the first update log record for a transaction

Then there's a series of update records (btree block splits have five records in string)

Then there are one or more commit log records, xor one or more abort log records (either commit or abort, never are both present)

Then, in the case of an abort, there is a set of undo log records and after that there are one or more forgotten log records to terminate the transaction history

# Database consistency (database must be serialized wherever it goes) ...

The big thing about transaction histories in the log is that even though they are interspersed in order, you must never have the case where a log record touching data for one transaction is interspersed with a log record touching the same data for another transaction

You must be able to sort the entire log by transaction timestamp to the same effect as the original log when replayed back into the database at recovery time: yielding the holy grail of transaction systems, Sortable Transaction Histories

# Database consistency (database must be serialized wherever it goes) ...

And no matter how many nodes are involved in sharing transactions, if all the logs for those nodes are merged they should yield a joined log with joint serialized histories for all the transactions anywhere in the entire contiguous computational universe of NSK systems (which is why we do two phase commit coordination between nodes)

And if the log is shipped to an RDF backup node, since it is serialized and transmitted sequentially, the result on the backup node is ***guaranteed*** to be the same as on the primary node.

# Database consistency (database must be serialized wherever it goes) ...

I is for Isolation and should probably be spelled ACLD, because isolation in database means locking

For NS SQL (and Enscribe), transaction duration locks are either update locks which block reads and updates, or shared read locks which block updates only

Locks are only released after all of their associated transaction database work has ceased, and once the totality of database changes have hit the log disk: the locks are the fingers of the correctly ordered database in the log reaching out to the live database and guaranteeing serialized behavior in the applications interacting through that database

# Database consistency (database must be serialized wherever it goes) ...

Because of transactional isolation:

- Applications communicate with each other using relational propositional logic (SQL queries) ...
- inserting, updating or deleting truthful propositions (records) ...
- in shared repositories of mutually agreed upon truths (SQL tables) ...
- through the database in complete, uninterrupted compound sentences (transactions)
- pausing in real-time, only to hear the complete, uninterrupted compound sentences of other concurrent applications (locks)
- otherwise, running at warp speed unhindered of any other blockage to performance (Minimum Latency)

# Database consistency (database must be serialized wherever it goes) ...

Finally, D is for Durability and means that once TMF says it's done, it's done

If the ENDTRANSACTION procedure call returns FEOF, and one nanosecond later the system crashes, the data is there

For the proposed ISC system, that means it's done on the primary node and the backup node, for the same extremely available database

For an RDF system, that means it's done on the database on the primary node, and after all the audit reaches the backup node, it's done there, too

# So What's The ISC About?

How is that name not a lie?

- ZLE
- Not Eternal ISC
- ISC for one observer's lifespan (IMHO)

What about networks?

- Inside ISC
- Outside ISC

What is the point of architectural focus?

- Moving finger writes and having writ moves on ...



i n v e n t