APPROACHES TO FAULT TOLERANCE

jim gray
Tandem
19333 Vallco Parkway
Cupertino Ca. 95014
tel: 408-725 6212
summer 1984

OVERVIEW

* PRAGMATICS

* SYSTEM ARCHITECTURE

* FAULT TOLERANT EXECUTION
  (PROCESS PAIRS)
  TRANSACTIONS

* FAULT TOLERANT STORAGE
  (REPLICATED DATA)
PRELIMINARIES

MTBF: Mean Time Between Failures

MTTR: Mean Time To Repair

RELIABILITY: DOESN'T DO THE WRONG THING
   ALSO MTBF

AVAILABILITY: DOES THE RIGHT THING
   ALSO = MTBF/(MTBF+MTTR)

TYPICAL NUMBERS

MODULES: MTBF 10Khrs (about a year)

FAIL-FAST: WORK CORRECTLY or
   ANNOUNCE FAILURE

LONG HAUL LINES: HARD MTBF 4Khrs
   HARD MTTR 10Hrs
   TRANSIENT MTBF 10Minutes
   TRANSIENT MTTR >1Sec (spike)

GOOD SINGLE FAULT TOLERANT HARDWARE

MTBF FOR SYSTEM HARDWARE > 10 YEARS
TYPICAL NUMBERS

ATM EXAMPLE AVAILABILITY:
ATM 97% (Broken, PM, out of money)
LINE 99% (100 min/week)
HOST 99.9% (10 min/week)

CENTRAL SYSTEM IS NOT PROBLEM.

DISCUSSION HERE FOCUSES ON HOST SYSTEM

REAL PROBLEMS: REMOTE MAINTENANCE
NETWORK MANAGEMENT
CHANGE CONTROL

NOT DISCUSSED HERE

WHY DO FAULT TOLERANT SYSTEMS CRASH?

PEOPLE MOSTLY
SOFTWARE OFTEN
HARDWARE RARELY

CONCLUSION:

BUILD SYSTEMS SIMPLE TO OPERATE/MAINTAIN
FIND WORKABLE APPROACH TO SOFTWARE FAULTS
WHY DO WE WANT HIGH AVAILABILITY?

ANATOMY OF A CRASH:
  DUMP (10 min)
  OS RESTART (5 min)
DB RESTART (5 min)  DC RESTART (30 min)

45 min!

PLUS HUMAN RESTART TIME!

CONCLUSION:

WITH LARGE NETWORKS MUST MAINTAIN SESSIONS
LARGE NETS REQUIRE HIGH AVAILABILITY
HENCE WANT FAULT TOLERANCE

FAULT TOLERANT ARCHITECTURE

* FAIL FAST MODULES (SOFTWARE AND HARDWARE)
* EXTRA CAPACITY (TOLERATE FAILURE)
* MULTI-COMPUTER (NO SHARED MEMORY)
  => FAULT ISOLATION
* MESSAGE BASED OS FOR COMMUNICATION
  ACROSS FIRE-WALLS
* PROCESS PAIRS TO TOLERATE FAULTS
* SESSION ORIENTED COMMUNICATION SO
  * DETECT LOST-DUPLICATE MESSAGES
  * CAN TALK TO PROCESS PAIR
  * CAN USE TRANSACTIONS
THE CASE AGAINST SHARED MEMORY

* STRONGEST ARGUMENT IS HISTORY
  EXPERIENCE OF ATT, IBM, HONEYWELL, ...

* MTBF(N) = MTBF OF AN N PROCESSOR
  SHARED MEMORY SYSTEM

  THEN
  MTBF(N) <= (MTBF(1))/N

*WHY? NO ONE KNOWS BUT:
  * POOR FAULT ISOLATION
  * MORE AND FANCIER SOFTWARE
  * RACE CONDITIONS
  * FASTER INSTRUCTION RATE

* FREE ADVICE: AVOID SHARED MEMORY

SESSIONS

* OPEN { WRITE | READ }* CLOSE
  * HIDES PATH FailURES

* EACH MESSAGE HAS A SEQUENCE #

* SEQUENCE NUMBERS DETECT
  * LOST MESSAGES
  * DUPLICATE MESSAGES
  * TRANSACTION COMMIT TIED INTO
    SESSION MANAGER.

* SESSION TO PROCESS PAIR SWITCHES
  TO BACKUP IF PRIMARY FAILS.
PROCESS PAIRS

TWO PROCESSES:
PRIMARY: DOES ALL THE WORK
BACKUP: PASSIVE, BUT CONTINUES CONVERSATION IF PRIMARY FAILS.

THREE KINDS OF PROCESS PAIRS:

* LOCKSTEP: REPLICALED EXECUTION
* SHADOW: BACKUP PASSIVELY TRACKS PRIMARY
  (may be a few messages behind)
* PERSISTENT: BACKUP RESETS ON TAKEOVER
  (may create it on takeover)

PROCESS PAIRS: THE CASE AGAINST LOCKSTEP

KNOWN FACT: MOST HARDWARE FAULTS ARE SOFT
TRANSIENT RATIO IS 5:1 OR 100:1

CONJECTURE: FOR TESTED SOFTWARE
MOST SOFTWARE FAULTS ARE SOFT

WHY?
EXPERIENCE:
PROCESS PAIR TAKEOVER FAILS < 1:100.

ARGUMENT:
MOST THINGS WORK BUT STRANGE ONES DON'T
(RACE CONDITIONS)
BACKUP HAS SLIGHTLY DIFFERENT STATE SO
TAKES DIFFERENT PATH UNLESS LOCKSTEP

CONCLUSION: AVOID LOCKSTEP PROCESS PAIRS
PROBLEMS WITH PROCESS PAIRS

* CONJECTURE => AVOID LOCKSTEP

* SHADOW => HARD TO PROGRAM

* PERSISTENT => LOST STATE

SALVATION:

* TRANSACTIONS ALLOW PERSISTENT PROCESSES TO GIVE RELIABILITY AND AVAILABILITY. TRANSACTIONS CLEAN UP DATA AND SESSION REF TANDEM PATHWAY = TMF (TM).

* KERNEL-LEVEL PROGRAMMERS (DEVICE DRIVERS) ARE BELOW THE TRANSACTION LEVEL (SIGH). SO STILL NEED SHADOW PROCESS PAIRS.

MORE SALVATION FOR PROCESS PAIRS?

* AUTOMATIC SHADOWS (ref: Borg ACM SOSP 83)
  * AUTOMATIC MESSAGE SEND TO BACKUP
  * AUTOMATIC STATE COPY TO BACKUP
  * PREDICTION:
    * TRANSACTIONS MAKE SHADOWS IRRELEVANT EXCEPT FOR KERNEL.
    * AUTOMATIC SHADOWS COST MUCH MORE THAN MANUAL (MESSAGES AND BYTES).

* TRENDS IN MANUAL SHADOWS:
  * SEND LOGICAL LOG OF STATE CHANGE BACKUP IS MORE ACTIVE. (Borr VLDS 84)
SOFTWARE FAULT TOLERANCE: TRANSACTIONS

PROGRAMMER'S MODEL:

BeginTransaction
Do
Do
Do
CommitTransaction | AbortTransaction

TRANSACTION IS ATOMIC: ALL OR NOTHING
DURABLE: EFFECTS SURVIVE CRASH etc.

SEE C.J. DATE Intro to DB Vol. 2.
WHAT ARE TRANSACTIONS GOOD FOR?

* TRANSACTIONS GIVE RELIABILITY
  NOT AVAILABILITY

* SIMPLIFY ERROR HANDLING

* PROTECT DATA AGAINST CRASHES

* ALLOW USE OF PERSISTENT PROCESSES
  INSTEAD OF SHADOW PROCESSES
  USED IN THIS WAY
  TRANSACTIONS GIVE AVAILABILITY

* REMEMBER WE WANT TO KEEP THE SYSTEM UP

REPLICATED DATA

* EXACT REPLICAS

  * RAWA (read any write all)

  * MIRRORS

* MAJORITY REPLICAS

* MASTER COPY

  * SNAPSHOTS

  * ASAP UPDATES
EXACT REPLICAS: RAWA

* HAVE N COPIES OF DATA

* RAWA (Read Any, Write All)

* IF A COPY IS DOWN FOR ME,
  IT MIGHT STILL BE UP FOR SOMEBODY
  (e.g. NETWORK PARTITION).

* HENCE SINGLE FAILURE =>
  * READS ALLOWED
  * WRITES DISABLED

EXACT REPLICAS: RAWA

* IMPROVES RELIABILITY

* IMPROVES READ AVAILABILITY

* REDUCES WRITE AVAILABILITY

* SUPPORTED BY MOST SYSTEMS (INDICES)

* TRICK: INDEX OF A FILE MAY BE
  REPLICA OF A FILE

* USED ONLY RARELY BECAUSE OF UPDATE
  RESTRICTIONS. MIRRORS USED MUCH MORE.
EXACT REPLICA: MIRRORS

* HAVE N COPIES OF DATA

* IF ANY COPY IS DOWN FOR ME,
  THEN I KNOW IT'S DOWN FOR EVERYBODY
  (NO PARTITIONING).

* ALL COPIES HAVE A VERSION #
  USE VERSION # TO DETECT ANTIQUES

* RAWA (read any write all available)

* USE FUZZY COPY TO ADD NEW VERSION.

EXACT REPLICA: MIRRORS

* MIRRORS GOOD IN LOCAL NET
  (or single system)

* IMPROVES AVAILABILITY AND RELIABILITY

* EXAMPLES ACP, IMS FastPath, Tandem
  Burroughs DMS2.

* MIRRORS BAD IN LONG HAUL NET
  BECAUSE NET FAILURE POSSIBLE

VIOLATES: "IF IT'S DOWN FOR ME
  IT'S DOWN FOR YOU".
EXACT REPLICAS: MAJORITY

* HAVE N COPIES EACH WITH A VERSION #

* PICK INTEGERS R and W SO THAT
  \[ R < W \quad \text{and} \quad R + W > N \]

* READ R COPIES AND TAKE MAX VERSION #

* WRITE W COPIES AND BUMP TO MAX VERS#

* Ref: Gifford, ACM SOSP 1979

---

EXACT REPLICAS: MAJORITY

* ALGORITHM TOLERATES N-MAX(R,W) DOWNS

* IMPROVES AVAILABILITY AND RELIABILITY

* ALGORITHM IS EXPENSIVE IF READS ARE COMMON.

* DOESN'T MAKE SENSE IN LOCAL NET (USE MIRRORS).

* IS EXPENSIVE IN LONG-HAUL NET (MUST READ AT LEAST R >1)

* I HAVEN'T SEEN ANYONE USE IT YET
MASTER COPY: SNAPSHOT

* HAVE A MASTER COPY OF THE FILE OR RECORD

* HAVE MANY SLAVE COPIES

* UPDATE THE MASTER ANYTIME

* UPDATE THE SLAVE PERIODICALLY (once a day, once a week).

  * FULL COPY (makes sense for records)

  * DELTA (makes sense for files)

  * LOG PROVIDES DELTA.

MASTER COPY: SNAPSHOT

* GIVES STALE DATA TO ALL BUT MASTER

* DOESN'T HURT UPDATE AVAILABILITY

* IMPROVES READ AVAILABILITY (STALR)

* WIDELY USED.

* MOST BANKS WORK ON

  * SNAPSHOT plus

  * MEMO POST plus

  * NIGHT BATCH RUN FOR NEXT SNAPSHOT
MASTER COPY: ASAP UPDATE

* HAVE MASTER COPY OF THE FILE OR RECORD
* HAVE MANY SLAVE COPIES
* UPDATE THE MASTER ANYTIME

* UPDATE THE SLAVE As Soon As Possible (once the transaction commits)

* WAIT FOR TRANSACTION COMMIT

* LAUNCH ASYNCHRONOUS TRANSACTION TO EACH COPY OR

* SPOOL REDO LOG TO EACH COPY

MASTER COPY: ASAP UPDATE

* GIVES STALE DATA TO ALL BUT MASTER

* IF ALL OK, DATA IS SECONDS OLD (not very stale).

* GOOD FOR DISASTER RECOVERY

* IMPROVES READ AVAILABILITY

* IMPROVES WRITE AVAILABILITY IN CASE OF DISASTER

* BEING USED INCREASINGLY
  Anderton & Norman: "Empact: a Distributed Manufacturing Application"
  Spring NCC 83 proceedings