N-VERSION PROGRAMMING: A FAULT-TOLERANCE APPROACH TO RELIABILITY OF SOFTWARE OPERATION

Liming Chen, Algirdas Avizienis
Presented by Yifei Li

Definition

- *N-version programming* is defined as the independent generation of N >=2 functionally equivalent programs, called "versions", from the same initial specification

General Idea

- A *driver* coordinates the execution of n versions and compare their correspondent results
- For n >= 3, a result is voted as a result of majority decision. For n = 2, the comparison algorithm checks if two results match
- For numerical computations, an allowable discrepancy range is used by the comparison algorithm
  - Agreeing version: its result falls into an allowable discrepancy range.
  - Otherwise, a disagreeing version

Special Mechanisms

- *Comparison vectors* (c-vectors)
  - C-variables: hold results to be compared with their counterparts from other versions
  - Comparison status indicators: if some significant events have occurred during the generation of c-variables

Special Mechanisms

- Synchronization mechanisms
  - Cross-check points (cc-points): points where c-vectors are generated and employed for comparison

Special Mechanisms/Synchronization mechanisms

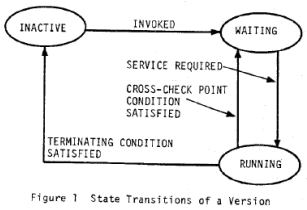


Figure 1 State Transitions of a Version

Comparison with Recovery-blocks

- Less storage overhead
- No special mechanisms needed to coordinate parallel processes
- Errors can be detected and recovered as early as possible
- No need to design acceptance test

An implementation of 3-version programming

```
VERSION¡:  PROCEDURE OPTIONS (TASK);
   DCL 1 C-VECTOR¡                    EXTERNAL,
         2 { comparison variables }
         ; { status      flags     };
   DCL (DISAGREE¡, GOODBYE)  BIT(1) EXTERNAL;
   DCL (SERVICE¡, COMPLETE¡) EVENT  EXTERNAL;
   DCL FINIS       BIT(1)    INIT('0'B);
   other declarations;
   DO WHILE (¬FINIS);
      WAIT (SERVICE¡);
      COMPLETION (SERVICE¡) = '0'B;
      IF ¬GOODBYE & ¬DISAGREE¡
      THEN CALL PRODUCE;
      ELSE FINIS = '1'B;
      COMPLETION (COMPLETE¡) = '1'B;
   END;
   PRODUCE:  PROCEDURE;
             produce C-VECTOR¡;
             END PRODUCE;
   END VERSION¡
```

Figure 2 A Schema for the i-th Version of Code

An implementation of 3-version programming

```
ACCEPTANCE:  PROCEDURE OPTIONS (MAIN);
   DCL VERSION¡     ENTRY;
   DCL 1 C_VECTOR¡  EXTERNAL,
         2 { comparison variables }
         ; { status      flag      } ;
   DCL DISAGREE¡     BIT(1)     EXTERNAL,
       GOODBYE       BIT(1)     EXTERNAL;
   DCL SERVICE¡      EVENT      EXTERNAL,
       COMPLETE¡     EVENT      EXTERNAL;
   other declarations;
   COMPLETION (SERVICE¡) = '1'B;
   COMPLETION (COMPLETE¡) = '0'B;
   CALL VERSION¡ TASK EVENT (FINIS¡);
   DO WHILE (need_more_service);
      WAIT (COMPLETE¡);
      COMPLETION (COMPLETE¡) = '0'B;
      process C_VECTOR¡;
      IF ¬need_more_service THEN GOODBYE = '1'B;
      COMPLETION (SERVICE¡) = '1'B;
   END;
   WAIT (FINIS¡);
   END ACCEPTANCE;
```

Figure 3 A Schema for an Acceptance Program

An implementation of 3-version programming

```
DRIVER:      PROCEDURE OPTIONS (MAIN);
   DCL (VERSION1, VERSION2, VERSION3) ENTRY;
   declare (C_VECTOR1, C_VECTOR2, C_VECTOR3);
   DCL (DISAGREE¡, DISAGREE2, DISAGREE3, GOODBYE)
       BIT(1) EXTERANL;
   DCL (SERVICE¡, COMPLETE¡,
        SERVICE2, COMPLETE2,
        SERVICE3, COMPLETE3) EVENT EXTERNAL;
   other declarations;
   initialize (SERVICE¡, COMPLETE¡) as in ACCEPTANCE;
   CALL VERSION1 TASK EVENT (FINIS1);
   CALL VERSION2 TASK EVENT (FINIS2);
   CALL VERSION3 TASK EVENT (FINIS3);
   DO WHILE (need_more_service);
      WAIT (COMPLETE1, COMPLETE2, COMPLETE3);
      process (C_VECTOR1, C_VECTOR2, C_VECTOR3);
      IF ¬DISAGREE1 THEN COMPLETION(COMPLETE1)='0'B;
      IF ¬DISAGREE2 THEN COMPLETION(COMPLETE2)='0'B;
      IF ¬DISAGREE3 THEN COMPLETION(COMPLETE3)='0'B;
      IF ¬need_more_service THEN GOODBYE='1'B;
      COMPLETION(SERVICE1)='1'B;
      COMPLETION(SERVICE2)='1'B;
      COMPLETION(SERVICE3)='1'B;
   END;
   WAIT (FINIS1, FINIS2, FINIS3);
   END DRIVER;
```

Inexact Voting

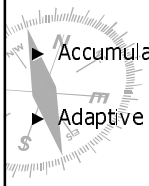
- Numerical computations may suffer from
  - Rounding errors
  - Unstable algorithms
- Thus, we can not require majority of correspondent results have exactly the same values

Inexact Voting/Adaptive Voting

- Adaptive voting:
  - R = W1*R1 + W2*R2 + W3*R3
  - W1, W2 and W3 are calculated dynamically and their sum should be 1
  - The goal is to favor acceptable results and minimize the effect of disagreeing results
- The performance of a scheme to compute weights is influenced by its *tolerance parameter*
  - a measure of the allowable noise level

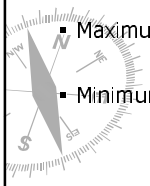
Inexact Voting/Adaptive Voting

- The tolerance parameter is difficult to determine
- The remaining effect of noise may not be acceptable
- Accumulation of residual effects of noise
- Adaptive voter implemented in software is slow



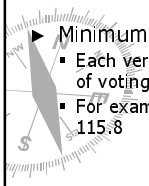
Inexact Voting/Non-Adaptive Voting

- Uses an allowable discrepancy range and differences of pairs of correspondent results in determining the voted result
  - Maximum(D12, D23, D31) <= ∫
  - Minimum(D12,D23,D31) <= ∫



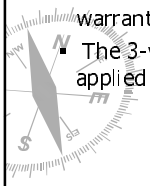
Inexact Voting/Non-Adaptive Voting

- ∫ is difficult to determine dynamically
- Maximum(D12, D23, D31) <= ∫
  - Too rigid
- Minimum(D12,D23,D31) <= ∫
  - Each version may have different effects on the outcome of voting
  - For example, ∫ = 0.9, R1 = 117.0, R2 = 116.5, R3 = 115.8



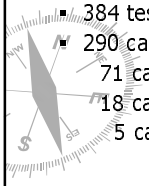
Experiment Results/MESS

- MESS (Mini-Text Editing System)
  - The methodology used to implement N-version programming is relatively simple
  - The effectiveness of 3-version programming warrant further investigation
  - The 3-version redundancy was successfully applied at subroutine level



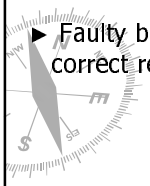
Experiment Results/RATE

- RATE, Region Approximation and Temperature Estimation
  - 7 programs implemented 3 algorithms
  - 12 3-version programs
  - 384 test cases
  - 290 cases contained no bad versions
    71 cases contained one bad version
    18 cases contained two bad versions
    5 cases contained three bad versions



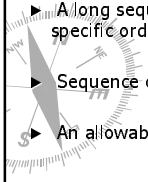
Experiment Results/RATE

- Other two versions that were running correctly could not proceed due to the errors introduced by one version
- Faulty but identical results may outvote correct results

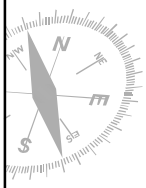


Conclusions

- System failures caused by performance limitations instead of functional problems
- No unique path to the solution of a problem
- A long sequence of outputs that can not be specified in a specific order
- Sequence of outputs is context dependent
- An allowable range of discrepancy is difficult to determine



Questions ?



Discussions

- What type of applications is N-version programming good for?