Aula 6 - Simulador

Monday, March 21, 2016 13:41

Revisão Adaptive DSD

Diagnóstico de Eventos

Definição de Evento: Um nodo falho fica sem-falha (recupera) ou um nodo sem-falha ifca falho (falha)

O diagnóstico completa quando todos os nodos sem-falha obtém info sobre o evento.

Latência: Número de rodada de testes a partir do momento em que o evento ocorre até completar o diagnóstico.

Teorema 1 & Colorário 1

(informal) Após 1 rodada de testes do algoritmo Adaptive DSD, os nodos sem-falha do sistema formam um anel.

Teorema 2

Dado o sistema S em uma determinada situação de falhas, após N rodada de testes, todo nodo sem falha i mantém $TESTED\_UP_i[x] = y$, para todo par de nodos sem-falha x e y tal que x testa y.

Prova

Escolha um nodo sem-falha arbitrário x.
Pela especificação do algoritmo, após uma rodada de testes, o $TESTED\_UP_x[x] = y$.
Após duas rodadas de teste, o nodo w testa x e obtém $TESTED\_UP_x$, fazendo:

$$TESTED\_UP_w[x] = TESTED\_UP_x[x] = y.$$

Nas i rodadas de teste subsequentes, mais i testadores obtém e atualizam localmente $TESTED\_UP[x] = y$.
O caminho mais longo em um ciclo do sistema S tem N nodos. Portanto, em no máximo N rodadas, todos os nodos atualizam $TESTED\_UP[x] = y$.

Teorema 3

Dado o sistema S com N nodos, cada um dos quais em um dos estados {falho, sem-falha} e após N rodadas de testes do algoritmo `Diagnose` executado em qualquer nodo sem-falha de S determina corretamente a situação de falhas do sistema.
Prova no artigo.

Qual o número máximo de testes executados em uma rodada *perfeita* (nodos aproximadamente sincronizados) do algoritmo Adaptive DSD? N!
Por quê? Cada nodo é testado *no máximo* uma vez. Ou seja, o Adaptive DSD é ótimo (optimal).

SMPL: Simulation Programming Language

Biblioteca da linguagem C

Tempo.c

```
#include<stdio.h>
#include<stdlib.h>
#include "smpl.h"
/* Eventos */
#define TEST 1
#define FAULT 2
#define REPAIR 3

/* descritor do nodo */
typedef struct {
     int id; // Identificador de facility SMPL
     // Outras estruturas locais são definidas aqui.
} Nodo;

Nodo *nodos;

/* Corpo do programa */
main(int argc, char *argv[]) {
     static int N; // Número de nodos do sistema
     static int token;
     static int event;
     static int r;
     static int i;
     static char fa_name[5];

     if (argc != 2) {
          puts("Uso correto: tempo [num-nodos]");
          exit(1);
     }
```

```
    N = atoi(argv[1]);
    smpl(0, "programa tempo");
    reset();
    stream(1); // 1 thread de execução

    // Inicialização dos nodos
    nodos = (Nodo*) malloc(N*sizeof(Nodo));
    for (i=0; i<N; i++) {
        memset (fa_name, '\0', 5);
        sprintf(fa_name, "%d", i);
        nodos[i].id = facility(fa_name, 1);
    }

    for (i=0, i<N, i++) {
        schedule (TEST, 30.0, i);
    }
    schedule (FAULT, 31.0, 2);
    schedule (REPAIR, 61.0, 2);

    while (time() < 100.0) {
        cause(&event, &token);
        switch(event) {
            case TEST:
                if status(nodos[token].id != 0)
                    break;
                printf("Sou o nodo %d, vou testar no tempo %
                5.1f\n", token, time());
                schedule(test, 30.0, token);
                break;
            case FAULT:
                r = request(nodos[token].id, token, 0);
                if (r != 0) {
                    puts("Não consegui falhar o nodo!");
                    exit(1);
                }
                printf("Sou o nodo %d, falhei no tempo %5.1f
                \n", token, time());
                break;
            case REPAIR:
                release(nodos[token].id, token);
                printf("Sou o nodo %d, recuperei no tempo %
                5.1f\n", token, time());
                schedule(TEST, 30.0, token);
                break;
        }
    }
}
```

Tarefas para o laboratório

0. Editar, compilar e executar este programa para N = 3, N=5

0. Editar, compilar e executar este programa para N = 3, N=5, ...
1. Cada nodo testa o seguinte do anel.
2. Cada nodo sem-falha executa testes, sequencialmente, até encontrar outro nodo sem-falha.
3. Cada nodo mantém localmente o vetor STATE[0..N-1] e atualiza as entradas correspondentes ao testar.
4. Cada nodo sem-falha, ao testar outro nodo sem-falha, atualiza adequadamente o vetor STATE (entradas de todos os nodos menos os que testou)