Johann Wolfgang Goethe-Universität
Frankfurt am Main

Fachbereich Informatik (20) Praktische Informatik *VSFT*

Das Fehlertoleranzkonzept von ATTEMPTO 2

R.Brause

Das Fehlertoleranzkonzept von ATTEMPTO 2 stellt eine Erweiterung und einen Ausbau des Fehlertoleranzkonzepts von ATTEMPTO 1 dar, das im Wesentlichen auf einer Replizierung der auszuführenden Tasks und anschließender dezentralen Fehlermaskierung und Voting aufbaut. Dabei ist die Größe einer zu replizierenden Task identisch mit dem auszuführenden Auftrag; mit dem Job als kleinster, zu replizierenden Einheit ist die Granularität der Fehlertoleranz ziemlich grob. In Abbildung 2 ist als Beispiel eine Kommunikationsstruktur für TMR gezeigt.

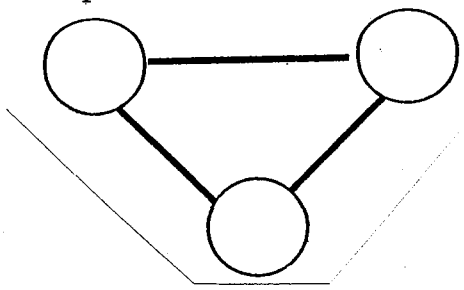

Abb. 1 Eine fehlertolerante TMR Struktur in ATTEMPTO 1

Demgegenüber sieht das Arbeitsmodell der Parallelisierung im ATTEMPTO 2- System [LUT1],[LUT2] ein relativ feinkörnige Aufteilung der Gesamtarbeit vor. Ausgehend von einem Hauptprogramm (*Wurzel*) wird parallel verteilbare Arbeitsteile auf anderen Prozessen anderer Prozessoren (*Unterknoten*) mit dem Modell des *Remote Procedure Call (RPC)* abgearbeitet. Diese Unterknoten können nun ebenfalls ihrerseits weitere Unterknoten mit parallel verteilbaren Teilen ihrer Teilarbeit beauftragen, so daß insgesamt die Auftragsstruktur einen Baum bildet. In Abbildung 2 ist ein solches Beispiel gezeigt.

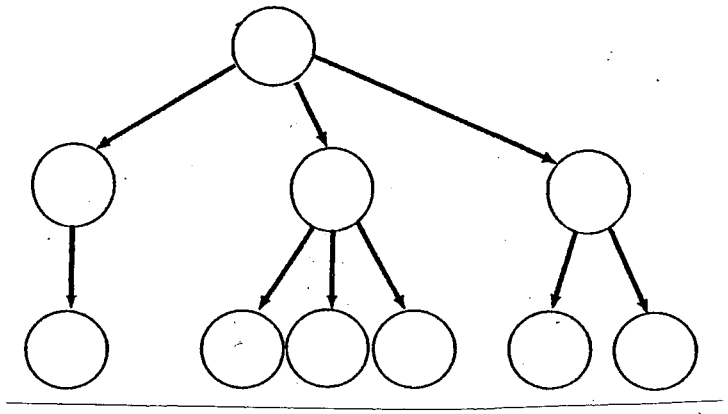

Abb. 2 Modell der Auftragsvergabe in ATTEMPTO 2

Wie könnte nun ein Fehlertoleranzmodell, besonders hinsichtlich eines Multiprozessornetzes mit vielen Prozessoren, wie es beispielsweise das Transputer-System des Lehrstuhls des IMMD darstellt, aussehen? Dabei müssen wir folgende Fakten beachten:

- Eine volle Replikation ist in vielen Systemen aufgrund von Speicherbeschränkungen nicht möglich. Beispielsweise haben Transputersysteme mit vielen Transputern meist wenig Speicher pro Transputer (ca 1MB) und keine Möglichkeit, elegant Prozeßteile (z.B. durch swapping oder paging) auf externe Massenspeicher auszulagern.

- Ausfälle passieren nicht so häufig im System, daß man immer nur im fehlermaskierenden Betrieb (ATTEMPTO 1) arbeiten müßte.

- Andererseits sollte schon gewährleistet sein, daß nach langer Arbeitszeit (z.B. langwierigen Berechnungen) nach einem Auftreten (Fail-save!) und Diagnose eines Hardware-Fehlers ("gutartige,eingrenzbare Fehler") der Job mit den letzten Teilergebnissen neu aufgesetzt und weiterrechnen kann.

Aus diesen drei Annahmen lassen sich folgende drei Schlußfolgerungen ziehen:

1) In ATTEMPTO 2 sollte es möglich sein, unterschiedliche Prozeßcode-Pakete zu schnüren, die nur den zur Ausführung eines Auftrags nötigen Programmcode enthalten. Das Konzept, auf allen Prozessoren den gesamten Programmcode aller möglichen Aufträge zu halten (*E-Prozeß-Konzept*) ist bei beschränktem Speicher nicht ratsam.

2) Eine Replikation eines Auftragnehmers für Fehlertoleranzzwecke findet nicht statt. Stirbt der Auftragnehmer (Fail-Save!), so beauftragt der Auftraggeber (*Manager*) nach Test und Diagnose einen neuen Prozessor mit den alten Daten.

3) Nur die obersten Auftraggeber sind repliziert und bilden eine Auftrags-Oligarchie.

Dieses Konzept aus 2) und 3) läßt sich durch eine Kombination von Abbildung 1 und 2 verdeutlichen; die sind in Abbildung 3 gezeigt. Hierbei bilden alle Kopien des Auftraggebers eine fehlertolerante Auftrags-Oligarchie, wobei nur einer davon tatsächlich die Aufträge vergibt. Die anderen überwachen seine Aktionen nach dem ATTEMPTO-1 Mechanismus für fehlertolerante Ausgabe [DAL].

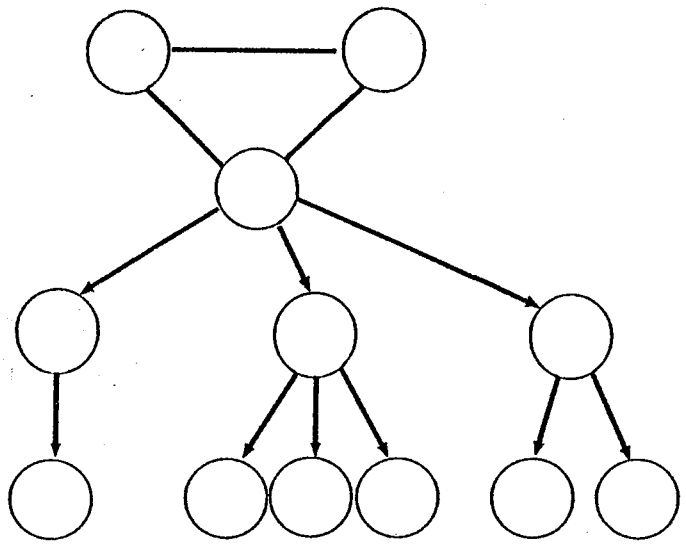

Abb. 3 Fehlertolerante, hierarchische Auftragsstruktur

Eine andere Möglichkeit besteht darin, die Aufträge unter den Einheiten der Oligarchie aufzuteilen, die wiederum jede für sich Untereinheiten mit Unteraufträgen beschäftigen. In Abbildung 4 ist eine solche Auftragsstruktur zu sehen.

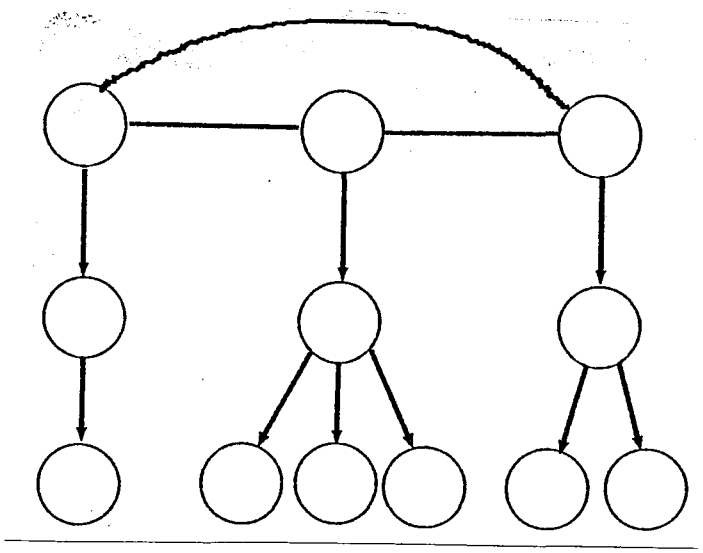

Abb. 4 Auftragsverteilte Oligarchie

Betrachten wir das Auftrags- und Fehlertoleranzkonzept 2) und 3), dessen Struktur in Bild 3 angedeutet ist, so gibt es dafür bereits ein Vorbild in der Literatur. In [WITT] entwickelten A. van Tilborg und D.Wittie für ihr MICROS-System die Idee einer fehlertoleranten Oligarchie und stellten in [TIL] ein Auftrags-Konzept unter dem Namen "Wave-Scheduling" vor. Dabei liegt die Vorstellung zugrunde, daß die Aufträge "wie eine Wellenfront" einheitlich durch die Ebenen des Baumes bis in die "entferntesten" Prozessoren (Blätter) vordringen und die Ergebnisse ebenso "wie eine reflektierte Welle" zurück zum Auftraggeber gelangen können.

Mit den vier Voraussetzungen

1) Die Prozessoren sind alle gleich (*Homogenität*)
2) Die Aufträge sind einheitlich (gleiche Länge, *unit tasks*); jeder Prozessor (Knoten) kann nur eine ganze Zahl von tasks gleichzeitig ausführen
3) Der Scheduler weiß im voraus nicht,
   - wieviel tasks bei einem schedule zu bewältigen sind
   - was jede task an resourcen benötigt
   - wie groß die Inter-task Kommunikation zwischen zwei tasks ist
4) Die Betriebssystem-Auftragsstruktur (Scheduler, Dispatcher) entspricht der Abbildung 4. Die Blätter des Baums sind Arbeiter, alle anderen Knoten Manager, die die Verteilung und lokale Resourcenbeschaffung übernehmen.

zeigten die Autoren, daß das verteilte Wave-Scheduling-Verfahren Resultate (Auslastung) bringt, die vergleichbar sind denen eines zentralen Scheduling. Dabei wird auch (im Unterschied z.B. zum Diffusions-Scheduling) ein statischer Deadlock vermieden.

Da die auftragsverteilte Oligarchie aus Abbildung 4 durch die dezentrale Zerteilung der Aufträge sich schwer in das baumartige Parallelisierungskonzept (*farming*) von ATTEMPTO-2 einfügt, wird stattdessen die fehlertolerante, auftraggeber-überwachte, homogene Auftragsstruktur aus Abbildung 3 für das Fehlertoleranzkonzept von ATTEMPTO 2 gewählt.

Das Grundkonzept enthät dabei zwei unterschiedliche Fehlertoleranzmechanismen. Für die oberste Oligarchie gilt das Konzept von ATTEMPTO 1, also eine Redundanz in der Arbeitsausführung und anschließender Diagnose für das Ergebnis. Durch eine Kennzeichnung der Auftragsvergabe als "globale Resource" wird automatisch im System der Vergleichsprozeß

angestoßen. Die Auftraggeber überprüfen sich also über den Signaturvergleich der Auftragsnachrichten. Die Ergebnisse müssen anschließend von den beauftragten Prozessoren als Nachrichten in einem "Broadcast" Modus an den replizierten Auftraggeber zurückgegeben werden.

Alle Unteraufträge (Auftragsverzweigungen) bekommen andere Fehlertoleranzmechanismen. Da die Speicherung von Zwischenergebnissen durch die Oligarchie gesichert ist, muß für die Ausführung der eigentlichen Arbeit nicht mehr hohe, fehlermaskierende Redundanz vorgesehen werden. Stattdessen recht es aus, mit den Fehlertoleranzforderungen 2) und 3) systematisch Rücksetzpunkte bei der Auftragsvergabe zu schaffen. Dazu bietet sich der Mechanismus der Auftragsvergabe geradezu an: bei der Auftragsvergabe werden alle Informationen in Form von Datenstrukturen und Puffern aufgehoben. Werden die Ergebnisse vom Beauftragten Prozessor zurückgeschickt, so werden die alten Funktionsparameter mit den neuen Werten überschrieben und der Rücksprung in die aufrufende Prozedur eingeleitet. Bemerkt dagegen der Auftraggeber durch Zeitüberwachung (Time-Out) des Auftrags oder durch ausbleibende Kontrollnachrichten (Totmann-Schalter oder Heart-beat) den Abbruch des Auftrags, so sind alle Informationen noch vorhanden, um ihn erneut aufzusetzen.

Allerdings verlangt ein solches Konzept, daß die Auftragnehmer-Prozessoren *Fail-save* Eigenschaft haben. Dies ist durch die üblichen Fehlererkennungsmechanismen, wie Selbsttests, Floationg-Point Error, Memory Parity Check etc. nicht immer gewährleistet. Will man eine verläßliche Fehlertoleranz auch bei funktionellen Fehlern, so kommt man um eine Funktionsüberwachung, beispielsweise durch einen zweiten Prozessor, nicht herum. Um die Fail-save Eigenschaft aller beteiligten Prozessoren sicherzustellen, ist es also nötig, jeweils die Auftragsvergabe mindestens zu duplizieren. Im fehlertoleranten Betrieb vergibt jeder Auftraggeber zwei Aufträge und vergleicht anschließend die Ergebnisse (Vergleichstest) miteinander. Besteht ein Unterschied, so muß der Auftrag erneut vergeben werden, anderfalls werden die Ergebnisse akzeptiert.

Das Fehlertoleranzkonzept von ATTEMPTO 2 benutzt also im Wesentlichen *roll-back* ähnlich wie im Tandem-System, wobei die Daten der Rücksetzpunkte durch die Auftragspufferung im Auftraggeber-Leichtgewichtsprozeß des RPC automatisch und benutzertransparent vorhanden sind.

Literatur

[DAL] M.Dal Cin, R.Brause, E.DIlger, J.Lutz, T.Risse
ATTEMPTO- An Experimental Fault-Tolerant Multiprocessor System
Microprocessing and Microprogramming Vol20, pp 301-308, North-Holland 1987

[LUT1] J.Lutz, R.Brause, M.DalCin, Th.Philipp
Ein Parallelisierungskonzept für ATTEMPTO 2
Interner Bericht 1/89 des Fachbereichs Informatik der J.W. Goethe Universität Frankfurt a.M., 1989

[LUT2] J.Lutz, R.Brause, M.DalCin, K.Mamers, Th.Philipp
Die Erweiterungen der ATTEMPTO 2 Laufzeitbibliothek
Interner Bericht 2/89 des Fachbereichs Informatik der J.W. Goethe Universität Frankfurt a.M., 1989

[TIL] A. van Tilborg, L.Wittie
Wave Scheduling: Distributed Allocation of Task Forces in Network Computers
IEEE Proc. 2th Int. Conf. on Distr. Comp. Systems, Paris 1981

[WITT] L.Wittie, A. vanTilborg
MICROS, A Distributed Operating System for MICRONET, A Reconfigurable Network Computer
IEEE Trans. on Comp. Vol C-29, No 12, Dec. 1980