Fehlertoleranz in verteilten Realzeitsystemen *

Quan Sun Horst Langendörfer

Institut für Betriebssysteme und Rechnerverbund
Technische Universität Braunschweig
Bültenweg 74/75, 38106 Braunschweig, Germany
Email: sun@ibr.cs.tu-bs.de

Zusammenfassung

Verteilte Realzeitsysteme werden in vielen zeitkritischen Anwendungen eingesetzt. Fehlertoleranzverfahren dienen der Steigerung der Zuverlässigkeit bzw. der Verfügbarkeit von Systemen. Traditionelle Methoden zur Unterstützung von Fehlertoleranz in verteilten Systemen erweisen sich für Realzeitsysteme wegen der Zeitbedingungen als unzufriedenstellend. In der vorliegenden Arbeit werden einige wichtige Aspekte beim Entwurf von fehlertoleranten verteilten Realzeitsystemen und weiter zu untersuchende Forschungsarbeit (im Allgemeinen) vorgestellt.

1 Einleitung

Verteilte Realzeitsysteme werden heute in einer großen Vielfalt von Anwendungen eingesetzt, so z.B in den Flugzeugsteuerungs-, Platzreservierungs-, Multimediasystemen und intelligenten Netzwerken. Diese Systeme arbeiten unter bestimmten Zeitbedingungen und sollten beim Auftritt von Fehlern weiter korrekt arbeiten. Durch Einsatz von Fehlertoleranzverfahren können die Zuverlässigkeit und Verfügbarkeit der Systeme erhöht werden. Einige existierende fehlertolerante Realzeitsysteme sind beispielsweise SIFT [1], MAFT [2], MARS [3], Delta-4 XPA [4], HARTS [5].

Eine wichtige Methode, Fehlertoleranz zu erreichen, ist die Einführung von Redundanz. Mögliche Redundanz ist: (1)Hardwareredundanz - mehrfache Hardwarekomponenten, z.B. Prozessoren, Speicher. (2)Softwareredundanz, z.B. die Realisierung einer Systemfunktion durch mehrere Prozesse, von denen jeder für sich allein die Funktion erbringen kann. (3)Informationsredundanz, z.B. Paritybits, Hamming-Fehlerkorrekturcodes. (4)Zeitredundanz, z.B. Wiederholung der Ausführung von Prozessen.

Wegen Zeitbedingungen in Realzeitsystemen wird Zeitredundanz normalerweise nicht benutzt. Deshalb sind viele traditionelle Fehlertoleranzverfahren (z.B. Checkpointing und Rollback) nicht mehr geeignet. Der Entwurf von fehlertoleranten Realzeitsystemen ist eine große Herausforderung. Im folgenden werden einige wichtige Aspekte der Berücksichtigung beim Entwurf von fehlertoleranten verteilten Realzeitsystemen und weiter zu untersuchende Forschungsarbeit vorgestellt.

2 Systemkonfiguration

Ein verteites System besteht aus einer Vielzahl von Knoten, die mittels Nachrichten über ein Netzwerk miteinander kommunizieren. Wenn die relativen Rechengeschwindigkeiten der

*Vortrag in der GCI-Jahreskonferenz, 25.-26. März, 1995, Stuttgart

Knoten und die Übertragungszeit der Nachrichten im Netzwerk begrenzt sind, ist dieses System ein *synchrones System*. Wenn alle beide unbegrenzt sind, geht es um ein *asynchrones System*. In einem verteilten System ist das Fehlermodell der Knoten (Prozesse) von großer Bedeutung:

Unterlassungsfehler (Fail-silent): Fehlerhafte Knoten (Prozesse) senden keine Nachrichten mehr. Alle gesandten Nachrichten im Netzwerk sind Nachrichten von fehlerfreien Knoten.
Byzantinische Fehler (Fail-uncontrolled): Fehlerhafte Knoten (Prozesse) könnten weiter falsche Nachrichten senden.

Übereinstimmungsproblem (consensus problem) ist eines der wichtigsten Probleme in verteilten Systemen [6]. Ziel ist, daß alle korrekten Knoten (auch beim Auftritt von Fehlern) zu einem gemeinsamen Ergebnis gelangen. Es werden bewiesen:

(1) Ein asynchrones System kann sogar einen einzigen Knotenfehler nicht tolerieren [7].

Zum Glück erweisen sich reale Systeme meistens als synchron. Für ein synchrones System (mit Knotenanzahl N):

(2) Beim byzantinischen Fehlermodell (ohne Authentifikationsmechanismen) können f Knotenfehler toleriert werden, falls N > 3f [8].

(3) Beim Unterlassungsfehler-Modell können f Knotenfehler toleriert werden, falls
N > f+1 [9].

Das Unterlassungsfehler-Modell wird von den meisten fehlertoleranten verteilten Realzeitsystemen benutzt. Damit der Knoten beim Auftritt von Fehlern keine Nachrichten mehr sendet, werden unterschiedliche Maßnahmen getroffen. Z.B. wird in MARS regelmaßig self-checking im Knoten gemacht, hauptsächlich mit Hilfe vom Betriebssystem. In Sequoia [10] gibt es spezielle Hardware (Komparator und Uhr-Schaltung). In Delta-4 besteht jeder Knoten aus zwei Komponenten: Host und NAC(Network Attachment Controller), s. Abb.1.

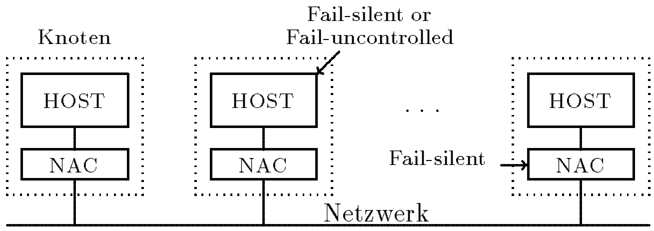

Abbildung 1: Delta-4 Hardware Architektur

Jede NAC-Komponente besteht aus zwei Prozessoren, die ein spezielles Protokoll ausführen, damit es gesichert wird, daß der Knoten beim Fehler keine Nachrichten senden wird. Eine ähnliche, aber bessere Methode findet man in VOLTAN [11], wo jeder Knoten aus mehreren Prozessoren besteht, die spezielle Protokolle ausführen.

Alle obig genannten Methoden haben aber Schwächen. Z.B. dürfen die beiden Prozessoren in Delta-4 nicht gleichzeitig fehlerhaft sein. In VOLTAN wird vorausgesetzt, daß das Rechnen der Prozessoren deterministisch ist (d.h. von einem gegebenen Anfangszustand werden gleiche Resultate für gleiche Eingabendaten bei keinem Fehlerauftritt ausgegeben). Weitere Untersuchungen zu diesem Thema sind nötig. Damit stehen allgemeine Fehlererkennung und -diagnose in einem unmittelbaren Zusammenhang.

3 Kommunikations-Unterstützung

Zuverlässige Kommunikation zwischen Knoten ist für verteiltes Rechnen von wesentlicher Bedeutung. Für fehlertolerante verteilte Realzeitsysteme ist die Gruppenkommunikation meistens notwendig (s.nächster Abschnitt). Ein *Gruppenkommunokationsprotokoll* (group communication protocol, multicast protocol) hat normalerweise folgende Eigenschaften [4][12]:

(1) *Übereinstimmung*
Jede Nachricht, die von einem fehlerfreien Knoten erhalten wird, wird auch von allen anderen *korrekten* Knoten der Gruppe erhalten. Die Lösung vom Übereinstimmungsproblem ist mit dem Knotenfehlermodell und der Zuverlässigkeit des Kommunikationsnetzwerks direkt verbunden. Für das Unterlassungsfehlermodell ist jede geschickte Nachricht im Netzwerk von einem fehlerfreien Knoten geschickt. Diese Situation vereinfacht enorm die Lösung des Übereinstimmungsproblems. Aber im Netzwerk könnten Nachrichten verloren gehen oder die Übertragungszeit kann manchmal sehr lang sein. Solche Situationen sollen sorgfältig berücksichtigt werden.

(2) *Reihenfolge*
Nachrichten werden von allen fehlerfreien Knoten der Gruppe in der gleichen Reihenfolge erhalten. Mit anderen Worten ist eine Nachricht entweder in gleicher Reihenfolge zu allen korrekten Knoten der Gruppe oder gar nicht geschickt. Mit dieser Eigenschaft spricht man von einem *atomaren Gruppenkommunikationsprotokoll* (atomic multicast protocol). Ein Protokoll nur mit der Eingenschaft der Übereinstimmung wird ein *zuverlässiges Gruppenkommunikationsprotokoll* (reliable multicast protocol) genannt.

(3) *Bekannte und begrenzte Protokollausführungszeit*
Das Protokoll soll auch beim Fehlerauftritt eine bekannte und begrenzte Ausführungszeit haben. Diese Eigenschaft ist für Realzeitsysteme am wichtigsten. Ein Protokoll mit dieser Eigenschaft wird *synchrones Protokoll* genannt, sonst handelt es sich um ein *asynchrones Protokoll*. Beispiele sind [12][13](für ein synchrones Protokoll) und [14][15][4] (für ein asynchrones Protokoll). Ein synchrones Protokoll braucht eine globale Uhr. Protokolle könnten auch ohne globale Uhr (z.B.[4]) beim Auftritt von Fehlern eine begrentzte Ausführungszeit haben. Doch ist für „harte"Realzeitsysteme eine globale Uhr notwendig. Verfahren zur Synchronisation physikalischer Uhren findet man z.B. in [16][17].

Die meisten existierenden Gruppenkommunikationsprotokolle erlauben nur relativ kleine Datenmengen, die zwischen Knoten pro Zeiteinheit übertragen werden. Aber für manche verteilte Realzeitsysteme (z.B. verteilte Multimediasysteme) sind die übertragenden Daten pro Zeiteinheit sehr groß. Obwohl Datenverluste oder -verfälschungen in solchen Systemen in beschränktem Umfang akzeptiert werden können, tritt ein neues Problem auf: Verzögerungsjitter [18][19]. Audio- und Videoströme sind kontinuierliche Datenflüsse, die aufeinanderfolgenden Dateneinheiten müssen mit einer spezifizierten Rate und mit begrenzten Schwankungen ausgeliefert werden. In den letzten Jahren wurden einige Transportprotokolle dafür entworfen [20][21], z.B. TP++(für Multimedia-Anwendung) [22], XTP(unterstüzt Massendaten- und Realzeitübertragung, Multicast und Transaktion) [23]. Trotzdem bleiben wichtige Fragen ungelöst, z.B. die Bereitstellung einer zuverlässigen Gruppenkommunikation, die Synchronisation von Datenströmen und effiziente gleichzeitige Unterstützung von Anwendungen wie Multimedia und verteiltes Rechnen.

4 Fehlertoleranzverfahren

Es gibt hauptsächlich zwei Fehlertoleranzverfahren für verteilte Systeme:

(1) Erstellung konsistenter Systemzustände regelmäßig und Zurücksetzung des Systems zu vorher erstellten Systemzuständen beim Fehlerauftritt (Checkpointing und Rollback).

(2) Replikation von Daten und Prozessen in verschiedenen Knoten.

Für das erste Verfahren wird das verteilte System als eine Menge von kooperierenden Prozessen angesehen. Ein globaler Systemzustand besteht aus den lokalen Zuständen der kooperierenden Prozesse und den Nachrichten, die schon gesandt aber noch nicht empfangen werden. Ein konsistenter globaler Systemzustand ist ein Zustand, der vom Systemanfangszustand erreicht werden kann. Zahlreiche Verfahren zur Wiederherstellung von konsistenten Systemzuständen in einer verteilten Umgebung sind schon vorhanden (s. [24] für einen Überblick).
Leider sind diese Verfahren für verteilte Realzeitsysteme meistens nicht geeignet. Erstens könnte Wiederherstellungszeit zu groß und unbestimmt werden. Zweitens sind die Daten in Realzeitsystemen unbeständig und verlieren mit der Zeit normalerweise ihren Wert, z.B. ist die Rücksetzung eines Systems für Flugverkehrskontrolle zu einem früheren Zustand vielleicht sinnlos und sogar falsch.

Replikation von Daten wird schon weitgehend untersucht. Trotzdem bleiben viele Probleme noch nicht gelöst (s. [25] auch für einen Überblick). Viele Replikationsprotokolle berücksichtigen hauptsächlich die Verfügbarkeit von Daten, ein Beispiel ist das Coda-System [26]. Für ein verteiltes Realzeitsystem muß aber die Zeitbedingung streng eingehalten werden. Normalerweise kann ein Realzeitsystem die Netzwerk-Partition nicht tolerieren. Ein Protokoll, das schwach konsistente Zustände von replizierten Daten im Realzeitsystem verwaltet, ist in vielen Situationen praktisch.

Das häufig benutzte Fehlertoleranzverfahren in verteilten Realzeitsystemen ist Replikation von Prozessen. In diesem Verfahren wird ein Prozeß in manchen Knoten repliziert. Alle Kopien von diesem Prozeß in Knoten im Netzwerk bilden eine Replika-Gruppe (jede Kopie wird ,,Replika"genannt, s. Abb.2). Es gibt hauptsächlich zwei Techniken für repliziertes Rechnen: aktive und passive Replikation.

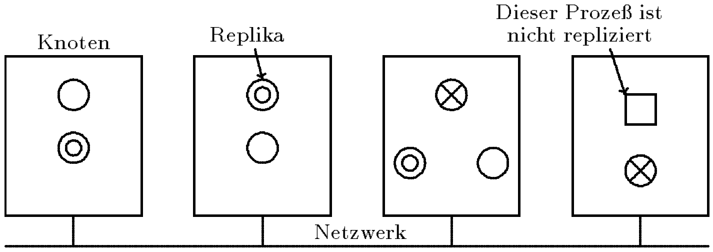

Abbildung 2: Replikation von Prozessen

Aktive Replikation (state machine approach [27][4][28]). Jede Replika in den fehlerfreien Knoten startet mit dem gleichen Zustand, verarbeitet die gleichen Input-Nachrichten und berechnet die gleichen Ausgaben. Das setzt voraus, daß jede Replika deterministisch ist und alle Input-Nachrichten in der gleichen Reihenfolge zu allen korrekten Knoten geliefert werden (dies kann durch ein atomares Gruppenkommunikationsprotokoll realisiert werden). Für Unterlassungsfehlermodell werden nur f+1 Replika (Kopien von Prozessen) gebraucht, um f Replikafehler zu tolerieren. Aber für byzantinische Fehler sind mindestens 2f+1 Replika notwendig, damit f Replikafehler toleriert werden können.

Passive Replikation (primary-backup approach [29][4]). Nur eine Replika von der Replika-Gruppe verarbeitet die Input-Nachrichten und ergibt die Ausgaben. Die Zustände von anderen Replika in der Gruppe werden von der rechnenden Replika durch Message-Passing geändert. Wenn diese Replika Fehler hat, kann eine von den anderen Replika die Arbeit weiter fortsetzen. Die Replika braucht nicht deterministisch zu sein und ein zuverlässiges Gruppenkommunikationsprotokoll ist schon genug.

Die Technik der aktiven Replikation hat mehr Overhead durch eine atomare Gruppenkommunikation. Die Technik der passiven Replikation braucht allerdings mehr Zeit, um beim Fehlerauftritt die Arbeit fortzusetzen. Obwohl schon viel Arbeit über Replikation von Prozessen gemacht wird, braucht man weitere Lösungen in Realzeitsystemen [30].

5 Schlußbemerkung

Wegen zunehmender Verbreitung der verteilten Systeme für Realzeitanwendungen wird der Einsatz von Fehlertoleranzverfahren in verteilten Realzeitsystemen immer wichtiger. Ein globaler Zustand in Realzeitsystemen muß den Zeit-Parameter umfassen. Redundanz-Management muß die Zeitbedingungen einhalten. Verfahren wie Checkpointing und Rollback sind meistens nicht geeignet. Schwach konsistente Protokolle für die Verwaltung von replizierten Daten und Prozessen sind normalerweise akzeptabel.

Außerdem ist Realzeit-Scheduling mit Redundanz-Management eng verbunden, auf diesem Gebiet bleiben noch Probleme offen [30]. Effiziente Behandlung von Common-Mode-Fehler (gleichzeitig hervorgebrachte Fehler in mehreren redundanten Kopien wegen eines einzigen Fehlers) ist auch eine große Herausforderung [31].

Literatur

[1] Melliar-Smith,P.M. and Schwartz,R.L., Formal Specification and Mechanical Verification of SIFT: A Fault-Tolerance Flight Control System, *IEEE Transactions on Computers*, C-31(7),pp.616-630, 1982

[2] Kieckhafer,R.M., Walter,C.J., Finn,A.M. and Thambidurai,P.M., The MAFT Architecture for Distributed Fault Tolerance, *IEEE Transactions on Computers*, 37(4),pp.398-405, 1988

[3] Kopetz,H., Damm,A., Koza,C., Mulazzani,M., Schwabl,W., Senft,C., Zainlinger R., Distributed Fault-Tolerant Real-Time Systems: The MARS Approach, *IEEE Micro*, 9(1),pp.25-40, 1988

[4] Powell,D., Delta-4: A Generic Architecture for Dependable Distributed Computing, ESPRIT Project 818/2252, vol.1, Springer-Verlag, 1991

[5] Shin,K.G., Kandlur,D.D., Kiskis,D.L., Dodd,P.S., Rosenberg,H.A. and Indiresan,A., A distributed real-time operating system, *IEEE Software*, Sept. 1992

[6] Turek,J., Shasha,D., The Many Faces of Consensus in Distributed Systems, *IEEE Computer*, 25(6):8-17, Juni 1992

[7] Fischer,M.J., Lynch,N.A., and Paterson,M.S., Impossibility of distributed consensus with one faulty process, *Journal of the ACM*, 32(2):374-382, 1985

[8] Pease,M., Shostak,R., Lamport,L., Reaching agreement in the presence of Faults, *Journal of the ACM*, 27(2):228-234, 1980

[9] Dolev,D. and Strong,H.R., Polynominal algorithms for multiple processor agreement, *Proc. 14th ACM Symposium on Theory of Computing*, pp.401-407, 1982

[10] Bernstein,P.A., Sequoia: A Fault Tolerant Tightly Coupled Multiprocessor for Transaction Processing, *IEEE Computer*, 21(2),pp.37-45, 1988

[11] Shrivastava,S.K., Ezhilchelvan,P.D., Speirs,N.A., Tao,S. and Tully,A., Principal Features of the VOLTAN Family of Reliable Node Architectures for Distributed Systems, *IEEE Transactions on Computers*, 41(5),pp.542-549, 1992

[12] Cristian,F., Dolev,D., Strong,R. and Aghili,H., Atomic Broadcast in a Real-Time Environment, in Fault-Tolerant Distributed Computing, Simons,B. and Spector,A. (Eds.), Springer-Verlag, 1990

[13] Babaoglu,O., Stephenson,P. and Drumond,R., Reliable Broadcast and Communication Models: Tradeoffs and Lower Bounds, *Distributed Computing*, No.2,pp.177-189, 1988

[14] Birman,K. and Joseph,T., Reliable Communication in the Presence of Failures, *ACM Transactions on Computer Systems*, 5(1),pp.47-76, 1987

[15] Chang,J.M. and Maxemchuk,N.F., Reliable Broadcast Protocol, *ACM Transactions on Computer Systems*, 2(3),pp.251-273, 1984

[16] Kopetz,H. and Ochsenreiter,W., Clock Synchronization in Distributed Real-Time Systems, *IEEE Transactions on Computers*, 36(8),pp.933-940, 1987

[17] Mills,D.L., A Kernel model for precision timekeeping, RFC 1589, Network Information Center, SRI International, März 1994

[18] Vetter,R. and Du,D.H., Distributed Computing with High-Speed Optical Networks, *IEEE Computer*, pp.8-18, Feb. 1993

[19] Zitterbart,M., Protokolle und Dienste für Hochleistungs-Kommunikationssysteme, in Praxisorientierte Parallelverarbeitung, Langendörfer,H. (Eds.), Carl Hanser Verlag, 1994

[20] Doeringer,W.A, Dykeman,D, Kaiserswerth,M., Meister,B.W., Rudin,H. and Williamson,R., A Survey of Light-Weight Transport Protocols for High-Speed Networks, *IEEE Transactions on Communications*, 38(11),pp2025-2039, 1990

[21] Biersack,E.W., Hochleistungs-Transportprotokolle, *it+ti*, pp.27-38, April 1993

[22] Biersack,E.W., Cotton,C.J., Feldmeier,D.C., McAuley,A.J. and Sincoskie,W.D., Gigabit Networking Research at Bellcore, *IEEE Networks*, 6(2),pp.42-49, March 1992

[23] Chesson,G., The Evolution of XTP, O.Spaniol and A.Danthine,Eds, *Proc. 3rd conference on High Speed Networking*, North-Holland publ., Amsterdam, The Netherlands, March 1991

[24] Deconinck,G, Vounckx.J, Cuyvers.R and Lauwereins.R, Survey of Checkpointing and Rollback Techniques. Bericht, Katholieke Uni. Leuven, Belgium, Juni 1993
(ftp: gate.esat.kuleuven.ac.be /pub/ACCA/FTMPS/REPORTS/f93-04.ps)

[25] Paris,J.F., The Management of Replicated Data, in Hardware and software Architectures for Fault Tolerance, Banatre and Lee.P.A (Ed.), Springer-Verlag, 1994

[26] Satyanarayanan,M., Kistler,J.J., Kumar,P., Okasaki,M.E, Siegel,E.H. and Steere,D.C., A Highly Available File Syatem for a Workstation Environment ,*IEEE Transactions on Computers*, 39(4),pp.447-459, 1990

[27] Schneider,F.B., The State Machine Approach: A Tutorial, in Fault-Tolerant Distributed Computing, Simons,B. and Spector,A. (Eds.), Springer-Verlag, 1990

[28] Birman,K.P., The Process Group Approach to Reliable Distributed Computing, Technical Report: TR-91-1216, Cornell University, 1991

[29] Budhiraja,N. and Marzullo,K., Tradeoffs in Implementing Primary-Backup Protocols, Technical Report: TR-92-1307, Cornell University, 1992

[30] Jahanian,F., Fault-Tolerance in Embedded Real-Time systems, in Hardware and software Architectures for Fault Tolerance, Banatre and Lee.P.A (Ed.), Springer-Verlag, 1994

[31] Lala,J.H. and Harper,R.E., Fault Tolerance in Embedded Real-Time Systems: Importance and Treatment of Common Mode Failures, in Hardware and software Architectures for Fault Tolerance, Banatre and Lee.P.A (Ed.), Springer-Verlag, 1994