Fehlertolerante Systeme

Autoren: Dr. Rüdiger Brause
Dr. Wolfgang Ehrenberger
Fritz Jörn
Hermann Johannes

Mit dem Einsatz von Rechnersystemen in den verschiedensten Bereichen steigt auch die Zahl der Anwendungen in sicherheitsrelevanten und -empfindlichen Sektoren. Die Anforderungen an die Zuverlässigkeit sowohl der Hardware als auch der Software nehmen zu. Seit die Firma Tandem vor zehn Jahren die ersten fehlertoleranten Systeme auf den Markt brachte, hat sich einiges geändert. Besoders Techniken zur Erstellung fehlerfreier, fehlertoleranter und robuster Software rücken heute in den Mittelpunkt vieler Diskussionen.

Aus der Sicht unserer Arbeitsgruppe an der Universität Tübingen (Leitung: Prof. Dr. M. Dal Cin) leiden die heutigen Produkte unter folgenden Schwachstellen:

- Die Fehlertoleranzmechanismen müssen vom Anwender in seinen Programmen berücksichtigt werden (kostspielige Programmänderungen!); Standardsoftware von dritter Seite ist deshalb kaum vorhanden
- Die erarbeitete Lösung ist meist sehr speziell, nicht modular und damit auch nicht portabel
- Die benutzte Hard- und Software entspricht keinem industriellen Standard
- Die Fehlertoleranz umfaßt meist nur Teile des Gesamtsystems
- Das System ist (relativ zur nicht-fehlertoleranten Version) einfach zu teuer

Im Gegensatz zu den Industrielabors war unsere Arbeitsgruppe in der Situation der Anwender. Als wir begannen, zur Erprobung unserer Fehlertoleranz-Konzepte den fehlertoleranten Rechner ATTEMPTO zu entwickeln, entschieden wir uns zu folgenden Vorgaben:

1) Die Fehlertoleranzmechanismen sollen für den Anwender transparent sein, d. h. alle, auch absolut gebundene Programme sollten ohne Änderung fehlertolerant ablauffähig sein.
2) Der Anwender kann entscheiden, wie er die Rechenleistung zwischen Fehlertoleranz und Durchsatz aufteilt.
3) Die Fehlertoleranzmechanismen müssen modular und hardwareunabhängig gewählt werden.
4) Die für die Fehlertoleranzmaßnahmen nötige Zusatzsoftware soll portabel sein.
5) Die Zusatzsoftware ist modular und strukturiert in einer höheren Programmiersprache zu schreiben.
6) Die Hardware-Komponenten müssen Standardteile, keine Spezialanfertigungen sein.
7) Die Software (Betriebssystem, Utilities) muß Standard sein.
8) Die Fehlertoleranz erstreckt sich bis auf die Ein- und Ausgabeleitungen.

Mit diesen Vorgaben ergeben sich folgende Design-Fragen:

● *Soll die Fehlertoleranz auf Gatter-Ebene, auf Chip-Ebene, auf Board-Ebene oder auf System-Ebene erfolgen?*

Auch ein mehrmaliges Abarbeiten und anschließender Vergleich der Ergebnisse auf demselben Rechner ist möglich („Zeitredundanz"). Im Gegensatz dazu benutzen Systeme, die nach einem Fehler bei einem früheren, geretteten Zustand wieder aufsetzen („Rollback"), zwar auch Zeitredundanz, ermöglichen aber nicht die Kontrolle und evtl. Korrektur der Ergebnisse. Diese Art von Fehlertoleranz ist somit nur bei unkritischen Anwendungen zu verwenden.

Die Fehlertoleranz auf Gatter-Ebene und auf Chip-Ebene setzt Spezial-Hardware voraus, die unserer Vorgabe 6 widerspricht. Da die zeitlichen Verfahren sehr aufwendig sind, entschieden wir uns für die Fehlertoleranz auf Board-Ebene; die kleinste ersetzbare Einheit ist also ein Single-Board-Computer

* Dr. R. Brause ist derzeit am Institut für Informationsverarbeitung der Universität Tübingen tätig.

Design eines Fehlertoleranten Rechners

Die Notwendigkeit fehlertoleranter Rechner ist heute unumstritten. Die zu erwartenden Marktanteile an der Gesamtproduktion von Rechnern sind groß. Die wenigen, bis jetzt vorhandenen oder neu erscheinenden oder angekündigten, industriellen Produkte weisen aber noch immer Mängel auf.

(SBC). Diese Entscheidung bringt auch weitere Vorteile: Die Fehlertoleranz-Mechanismen sind leicht übertragbar (Forderung 4), unabhängig von kleineren Modifikationen der Boards und beeinträchtigen nicht im Fehlerfall die Abarbeitungsgeschwindigkeit der Benutzerprogramme.

● *Sollen die Einheiten lose oder fest gekoppelt sein?*

In Abbildung 1 sind zwei Multi-Prozessorsysteme gezeigt.

Bei der festen Kopplung der Prozessoren in Abb. 1 a arbeiten alle Prozessoren mit einem gemeinsamen Speicherbereich, in dem das Betriebssystem und die Benutzerprogramme liegen. Damit der Systembus nicht die Rechenleistung empfindlich begrenzt, ist zusätzlich auf jedem Prozessor-Board ein schneller Cache enthalten. Diese Lösung hat folgende Nachteile:

- Bei Ausfall des Systembusses fällt das Gesamtsystem aus
- Defekte Prozessoren können die Daten der anderen korrumpieren und damit das System ebenfalls wertlos machen
- Standard-Programme (Lader, Debugger, etc.) müssen umgeschrieben werden, um Dateninkonsistenz zwischen Cache und Hauptspeicher zu vermeiden.

In dem lose gekoppelten System von Abb. 1 b sind diese Nachteile vermieden. Der Systembus wird nur noch zur Kommunikation (Austausch von Nachrichten zwischen den Prozessoren) verwendet; fällt er aus, so kann die Kommunikation auch über einen anderen Kommunikationsweg (z. B. den seriellen Bus in VME-Bus-Systemen) abgewickelt und dem Benutzer der Ausfall mitgeteilt werden. Das Betriebssystem und das jeweilige Anwenderprogramm sind für jeden Prozessor direkt zugreifbar, ohne damit die anderen Prozessoren (Rechnerleistung!) oder seine Daten zu tangieren.

Der Nachteil dieser Lösung ist allerdings auch evident: Wurden vorher zur Koordination der Prozessoren bei globalen Ressourcen (z. B. I/O-Einheiten) Monitor-Konstrukte im gemeinsamen Speicher eingesetzt, so geschieht nun die Koordination durch Nachrichtenaustausch. Dies benötigt mehr Rechnerleistung. Da sich aber Kommunikationsprotokolle sehr gut modular von den normalen Rechneraktivitäten abtrennen lassen, kann man den Nachteil relativ einfach durch einen besonderen Kommunikationsprozessor eliminieren.

● *Wie wird der Datenaustausch zwischen der Umwelt und dem Rechnersystem gestaltet?*

Bei der üblichen Lösung wird der Input von einem Interface empfangen und dann aktiv (DMA) oder passiv an die Prozessoren verteilt. Damit hängt aber wieder die Funktion des Gesamtsystems nur von einer einzigen Komponente ab. Das gleiche gilt auch für den Output.

Um dieses Problem zu vermeiden, werden im ATTEMPTO-System die Eingabedaten an jeden einzelnen SBC herangeführt (Abb. 2).

In diesem Fall kommen die Eingabedaten vom Benutzerterminal, dessen serielle Schnittstelle parallel mit allen seriellen Schnittstellen der SBC verbunden ist.

Bei besonders fehlertoleranten Prozeß-Systemen ist

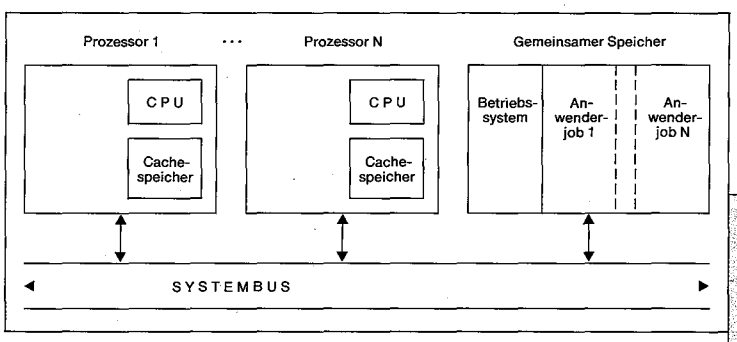

Abb. 1 a: Ein fest gekoppeltes Multi-Prozessor-System

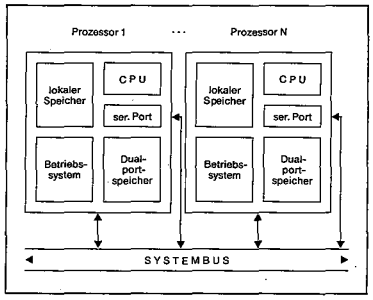

Abb. 1 b: Ein lose gekoppeltes Multi-Prozessor-System

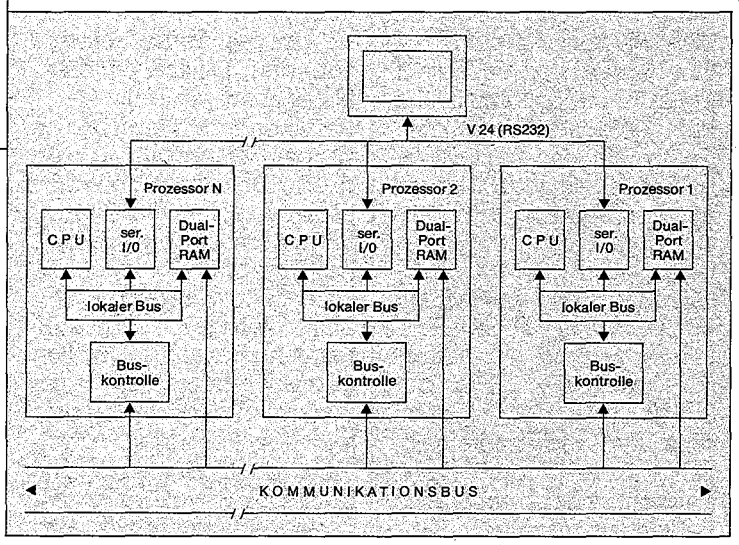

Abb. 2: Das ATTEMPTO-System

ein derartiger singulärer Datenbus sicher bedenklich. Abbildung 3 zeigt eine Alternative dazu, bei der Daten vom gleichen Prozeß, aber verschiedenen Sensoren ins System gehen und mit getrennten Leitungen wieder auf verschiedene Effektoren wirken.

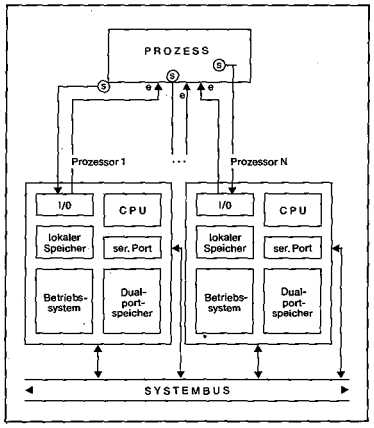

Abb. 3: Ein fehlertolerantes Prozeß-System

Überlagern sich diese Effektoren (Stellglieder) nur in der Wirkung, so kann für die Gesamtregelung auch der Ausfall eines Effektors toleriert werden.

Die Benutzersicht von ATTEMPTO

Der Benutzer sieht den Arbeitsplatzrechner als Single-User, Multi-Tasking-System; die Realisierung als Multi-Prozessor-System ist ihm verborgen. Das System bietet die Möglichkeit, die Fehlertoleranzeigenschaften im Gegensatz zu 3), 5) individuell für jede Anwendung zu wählen. Beispielsweise bedeutet ein Eintippen von „MEINPROGRAMM #2#", daß „MEINPROGRAMM" mit dem Fehlertoleranzgrad t=2 ausgeführt werden soll. Dies bedeutet, daß bei der Ausführung des Programms maximal t Defekte (und damit transiente oder permanente Fehler auf maximal t SBC) toleriert werden in dem Sinne, daß keine fehlerhaften Ergebnisse ausgegeben werden.
Der Benutzer kann das System soviele Programme gleichzeitig ausführen lassen, wie es die Systemkapazität erlaubt: mehrere Programme mit geringem oder wenige mit hohem Fehlertoleranzgrad.

Systemsoftware

Das Betriebssystem von ATTEMPTO besteht aus identischen, autonomen, lokalen Betriebssystemen, deren Kern von einem UNIX-ähnlichen Einprozessor-Mehrprozeß-Betriebssystem gebildet wird. Darauf baut eine Zwischenschicht auf, welche die Fehlertoleranzeigenschaften bereitstellt. Diese in Modula-2 programmierte Schicht besteht aus der Kommunikationsinstanz (CI) und der Fehlertoleranzinstanz (FTI) und ist für das Benutzerprogramm transparent. Vom Betriebssystemkern wird sie als normaler, hoch-prioritärer Prozeß behandelt.
An das Betriebssystem wird für die Inter-Prozessor-Koordination nur die Anforderung gestellt, daß Nachrichten in der gleichen zeitlichen Reihenfolge an die Fehlertoleranzschicht weitergegeben werden, in der sie an das Betriebssystem von den Device-Händlern übergeben werden.
Um Fehler tolerieren zu können, werden Kopien des Benutzerprogramms asynchron auf mehreren SBCs parallel abgearbeitet. Die Ergebnisse werden anschließend verglichen.
Zum jetzigen Zeitpunkt kommunizieren die lokalen Betriebssysteme nur zum Zwecke der Fehlertoleranz und der Resourcenverwaltung, doch läßt sich eine Funktions-Erweiterung zur Lastverteilung leicht vornehmen.

Wenn ein Benutzerprogramm eine Dienstleistung des Systems anfordert (z. B. Eingabe/Ausgabe), so geschieht dies durch einen System-Call an das zugrundeliegende Betriebssystem.

Die Kommunikationsinstanz (CI)

Die Kommunikationsinstanz bewirkt eine Umleitung der System-Calls zu der eingeschlossenen Zwischenschicht. Die CI teilt sich auf in einen dem Betriebssystemkern angehefteten Teil CIkernel, der die System-Calls in Nachrichten umformt und dem entsprechenden Teil CImain, der diese Nachrichten liest und der FTI zur fehlertoleranten Ausführung der Aktionen des Benutzerprogramms (lesen, schreiben, etc.) übergibt.

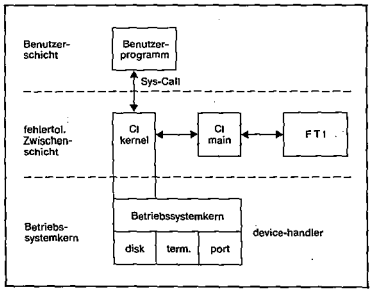

Abb. 4: Software-Konfiguration

Die Fehlertoleranzinstanz (FTI)

Die lokale FTI ist verantwortlich für folgende Betriebssystemaufgaben:
- lokale Interpretation der Benutzerkommandos an ATTEMPTO
- Management der systemweiten Resourcen (z. B. Terminal)
- Kontrolle der Daten von und zu den Ein- und Ausgabegeräten
- Systemweite Konsistenz der Systemtafeln

werdenFolgende Funktionen werden für Fehlertoleranzzwecke bereitgestellt:
- dezentrales Dispatching der Benutzerprogramme
- Vergleich der Ausgabedaten der lokalen Kopien eines Benutzerprogramms und anschließende Diagnose bei Nichtübereinstimmung
- Überwachung bei der Ausgabe der Daten
- Fehlerinterpretation und -behandlung

Aus Gründen der Fehlertoleranz besitzt jede FTI dazu ihre eigene Systemtafel, die alle aktuellen Angaben über Kollegen zur Programmausführung, anstehende Benutzerprogramme, Zustand der globalen Resourcen sowie aller SBC im System enthält.

Dispatching der Benutzerprogramme

Es findet kein Pre-Scheduling wie bei 5) und 3) statt. Stattdessen wird Dispatching nach dem Prinzip der Anziehung (attraction-principle) während der Laufzeit durchgeführt. Im Gegensatz zu der zentralen Hardware-Version dieses Prinzips in PLURIBUS 4) verwenden wir eine dezentrale Software-Version: Jeder freie Prozessor bewirbt sich um das in seiner Systemtafel als „noch zu bearbeiten" gekennzeichnete in der Eingabereihenfolge nächste Benutzerprogramm. Empfängt er eine Bewerbung für ein solches Programm, so trägt er den Bewerber dafür ein. Ist er selbst der Bewerber, so beginnt er mit der Ausführung des Programms. Alle Bewerbungen für bereits „besetzte" Programme werden ignoriert, außer der eigenen. Diese führt zur Bewerbung um das nächste freie Programm.
Vor jeder Ausgabeoperation vergleichen alle Prozessoren, die dasselbe Programm bearbeiten, die zur Ausgabe anstehenden Daten. Im Unterschied zu Stan-

dard-Mehrheitsentscheidungen muß bei unserer Fehlerdiagnose nicht jeder SBC alle Ergebnisse miteinander vergleichen. Stattdessen ist jedem Fehlertoleranzgrad ein sogenannter Testgraph zugeordnet, durch den festgelegt ist, welche Ausgaben zu vergleichen sind. Die Diagnose basiert auf den Ergebnissen dieser Vergleiche. Damit gewinnt jeder SBC Information über den Zustand der Kollegen.

Vermeidung von fehlerhafter Ausgabe

Die eigentliche Ausgabeoperation wird daraufhin von dem Prozessor durchgeführt, der als erster genügend Bestätigung erhalten hat. Die anderen intakten Kollegen überwachen diese Ausgabe.
Die Mechanismen zur Diagnose und Festlegung des ausgebenden Prozessors sind detaillierter in 1), 2) beschrieben.
Da wir primär transiente Fehler annehmen, ist es nicht ratsam, bei jedem auftretenden Fehler den SBC auszutauschen. Stattdessen führt jeder SBC eine eigene Fehlerfrequenzliste, in der auftretende Nichtübereinstimmungen der Resultate für spätere Auswertungen notiert werden.

Inter-Prozessor Koordination

Um einen Systemzusammenbruch bei Defekt des Systembusses oder des gemeinsamen Speichers zu verhindern, hat in ATTEMPTO jeder SBC seine eigene Systemtafel, die er durch Nachrichtenaustausch mit denen der anderen SBCs konsistent hält. Dabei tritt aber das Problem auf, daß die Verarbeitung einer Nachricht selbst wieder vom Zustand der Systemtafel abhängt; so kann beispielsweise ein Prozessor nur dann für einen Job in die Systemtafel eingetragen werden, wenn die Kollegenliste noch nicht vollständig ist. Eine Rückfrage führt in diesem Fall zu erhöhter Kommunikation und belastet das System zusätzlich. Deshalb wählten wir eine andere Lösung, nämlich die zeitliche Reihenfolge der Nachrichten zur Veränderung der Systemtafeln bei allen SBCs gleich zu halten. Damit sind bei gleichem initialem Ausgangszustand die Systemtafeln auch ohne Rückantwort jederzeit konsistent.
Genauer sind diese Mechanismen und die damit verbundenen Probleme in 6) beschrieben.

Referenzen

1) E. Ammann, R. Brause, M. Dal Cin, E. Dilger, J. Lutz, T. Risse ATTEMPTO: A Fault-Tolerant Multiprocessor Working Station, Design and Concepts, Proc. FTCS-13, Milano (1983) 10–13
2) M. Dal Cin, E. Dilger (Eds.), Self-Diagnosis and Fault-Tolerance, ATTEMPTO-Verlag, Tübingen 1981.
3) G. Färber, Task-Specific Implementation of Fault-Tolerance in Process Automation, in 2) 84–102.
4) D. Katsuki et al., PLURIBUS- an Operational Fault-Tolerant Multiprocessor, Proc. IEEE, Vol. 66/10 (1978) 1146–1159