Seminar

PG AutoLab

Verteilte Echtzeitsysteme

Sabrina Hecke

PG 522
Fachbereich Informatik
Technische Universität Dortmund
Lehrstuhl XII

21. bis 23. Oktober 2007

Inhaltsverzeichnis

1 Was sind Echtzeitsysteme?

Ein Echtzeit-Computersystem ist ein Computersystem, in dem die Korrektheit des Systems nicht nur vom logischen Ergebnis der Berechnung abhängt, sondern auch vom physikalischen Zeitpunkt, zu dem das Ergebnis produziert wurde.
Ein Echtzeit-Computersystem ist immer Teil eines größeren Systems. Dieses größere System wird Echtzeitsystem genannt.
Man unterscheidet zwei Typen von Echtzeitsystemen, und zwar gibt es zum Einen das zeitgesteuerte Echtzeitsystem und zum Anderen das ereignisgesteuerte System. Bei dem zeitgesteuerten Echtzeitsystem werden die Kommunikationsaktivitäten und Prozesse zu einem vorher festgelegten Zeitpunkt initiiert, was bei dem ereignisgesteuerten Echtzeitsystem durch einen signifikanten Zustandswechsel erfolgt.
Das Echtzeitsystem ist ein System, welches seinen Zustand als Funktion der Zeit verändert. Es wird in eine Menge von Subsystemen (s. Abbildung) zerlegt, welche Cluster genannt werden:

1. kontrollierte Cluster(kontrolliertes Objekt)
2. Berechnungs-Cluster(Echtzeitcomputersystem)
3. Operator-Cluster(Operator)

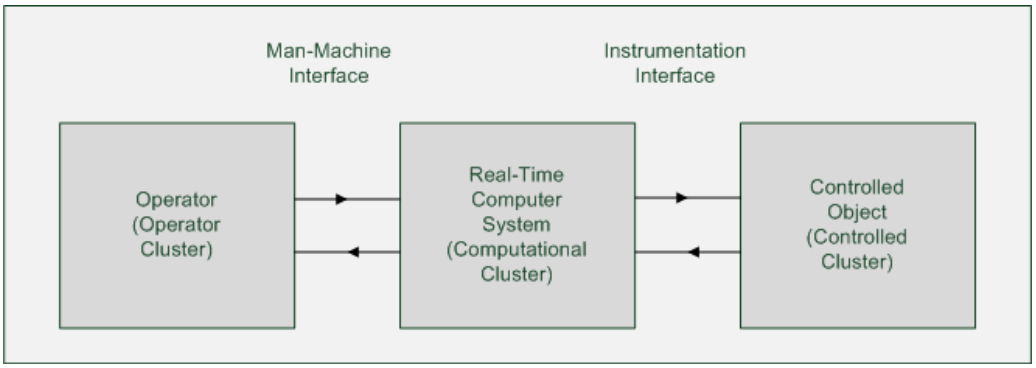

Abbildung 1: Echtzeitsystem

Das kontrollierte Objekt und der Operator bilden zusammen die Umgebung des Echtzeitcomputersystems. Die Schnittstelle zwischen menschlichen Operator und Echtzeitcomputersystem ist die Mensch-Maschine-Schnittstelle, welche aus Eingabe- und Ausgabegeräten besteht, wei z.B. Tastatur oder Monitor. Desweiteren gibt es eine weitere Schnittstelle zwischen Echtzeitcomputersystem und dem kontrollierten Objekt, die Geräteschnittstelle. Diese besteht aus Sensoren und Aktuatoren, welche physikalische Signale in eine digitale Form innerhalb des kontrollierten Objektes transformieren oder umgekehrt.

1.1 Echtzeittask

Innerhalb des Echtzeitcomputersystems führen nebenläufige Tasks die gewünschte Funktion aus. Ein Task ist die Ausführung eines sequentiellen Programms. Es beginnt mit dem Lesen der eingehenden Daten und dem internen Zustand eines Tasks, anschließend terminiert es mit der Erzeugung der Ergebnisse und dem Updaten des internen Zustandes.
Man unterscheidet drei Typen von Tasks nach dem Zeitpunkt seines Auftretens:

- **periodische Tasks**: Task wird alle T Zeiteinheiten(Periode) wiederholt ausgeführt
- **aperiodische Tasks**: Task tritt unregelmäßig und unvorhersehbar auf
- **sporadische Tasks**: Der zeitliche Abstand zwischen zwei Ausführungen eines Tasks ist begrenzt. Sporadische Tasks sind ebenfalls aperiodisch, da diese auch unvorhersehbar auftreten.

Echtzeittasks besitzen eine Deadline, d.h. es gibt eine maximale Zeit, in der die Ausführung eines Tasks beendet sein muss.
Zusätzlich wird der Echtzeittask in drei Kategorien unterteilt. Der Task heißt

- **weich**, wenn das Einhalten der Deadline zwar wünschenswert wäre, allerdings wenn die Deadline nicht eingehalten werden kann, führt dies zu keiner ernsthaften Beschädigung in der kontrollierten Umgebung und das Systemverhalten wird ebenfalls nicht gefährdet. Somit wären die Ergebnisse dieses Tasks weiterhin von Nutzen.
- **fest**, wenn das Ergebnis nach nicht Einhalten der Deadline keinen Nutzen mehr hat. Liegt das Ergebnis nicht innerhalb der Deadline vor, dann wird dieses zurückgesetzt.
- **hart**, wenn das nicht Einhalten der Deadline zu einer Katastrophe in der kontrollierten Umgebung führt.

1.2 Scheduling

In einem Echtzeitcomputersystem können nebenläufig mehrere Tasks ausgeführt werden, was so geschehen muss, dass alle zeitkritischen Tasks ihre Deadline einhalten können. Eine Menge von Tasks benötigt gleichzeitig Rechen- und Datenressourcen. Es existieren auch Ressourcen (z.B. CPU), die nicht zu einem bestimmten Zeitpunkt

mehrere Tasks ausführen können. Das Scheduling-Problem beschäftigt sich mit der Allozierung der Ressourcen, um Zeitanforderungen der Tasks zu erfüllen.
Es gibt verschiedene Scheduling-Algorithmen, die dieses Problem lösen:

1. **nicht-preemptives Scheduling**: Tasks werden solange ausgeführt bis sie abgearbeitet wurden.
2. **preemptives Scheduling**: Scheduler werden verwendet, wenn die Ausführungszeit eines Tasks zu lang ist. Dieser wird dann unterbrochen und ein anderer Task wird ausgeführt.
3. **dynamisch**: Prozessorallokationsentscheidungen finden zur Laufzeit statt.
4. **statisch**: Prozessorallokationsentscheidungen finden während der Kompilierung statt. Vor der Laufzeit wird Wissen über die Ankunftszeiten der Tasks, Ausführungszeiten und Deadlines mit in die Berechnung aufgenommen.

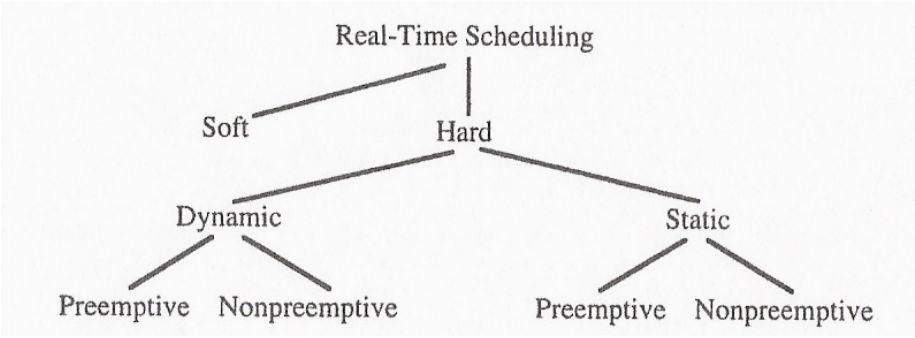

Abbildung 2: Scheduling

2 Verteilte Echtzeitsysteme

Ein verteiltes System besteht aus mehreren Knoten(Computern), die miteinander verbunden sind. Die Knoten kommunizieren durch Nachrichtenaustausch miteinander.

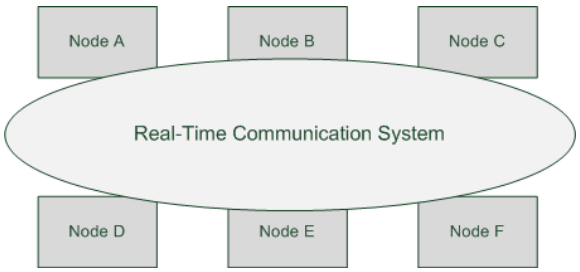

Abbildung 3: Verteiltes System

In einem verteilten Echtzeitsystem besteht das Berechnungs-Cluster aus mehreren Echtzeitcomputersystemen, die über ein Echtzeitkommunikationssystem miteinander verbunden sind.
Anforderungen an so ein System wären die Komponierbarkeit (Composability), Skalierbarkeit (Scalability) und Zuverlässigkeit (Dependability).

1. **Komponierbarkeit**: Um große Systeme designen zu können, zerlegt man das System zunächst in kleine Subsysteme und integriert diese zum Schluss wieder zu einem großen System.
   Kritisch ist dabei der zweite Punkt, nämlich das Integrieren der Subsysteme zu einem großen System, da die vorher spezifizierten und getesteten Eigenschaften nicht verloren gehen dürfen. Ein System, welches dieses garantiert, ist komponierbar.

2. **Skalierbarkeit**: In bestehenden Systemen müssen Funktionen geändert oder hinzugefügt werden können. Es heißt also skalierbar, wenn es offen gegenüber Veränderungen ist und es keine vordefinierte Einschränkung gegenüber Erweiterbarkeit hat.

3. **Zuverlässigkeit**: Auftretende Fehler müssen schnell behandelt werden. Dies geschieht durch Fehlertoleranz. Auf den Begriff Fehlertoleranz werde ich in dem Kapitel Fehlertolerante Systeme noch eingehen.
   Messwerte der Fehlertoleranz wären die Ausfallsicherheit (Reliability), Sicherheit (Safety), Wartbarkeit (Maintainability), Verfügbarkeit (Availability) und Sicherheit (Security).

- **Ausfallsicherheit**: Ist eine Funktion $0 \leq R(t) \leq 1$, dass das System korrekt während des Intervalls $[t_0, t]$ funktioniert unter der Annahme, dass das System zum Zeitpunkt t_0 korrekt arbeitete.
- **Sicherheit(Safety)**: Wahrscheinlichkeit $0 \leq S(t) \leq 1$, dass ein System entweder korrekt arbeitet oder seine Funktion auf eine Art und Weise beendet, so dass nicht die Funktionsweise anderer Systeme gestört oder Menschen gefährdet werden.
- **Verfügbarkeit**: Wahrscheinlichkeit $0 \leq A(t) \leq 1$,dass ein System zum Zeitpunkt t korrekt arbeitet.
- **Wartbarkeit**: Wahrscheinlichkeit M(t), dass ein fehlerhaftes System innerhalb einer Zeitdauer t repariert werden kann.
- **Sicherheit(Security)**: Fähigkeit eines Systems in Hinblick auf Datenschutz und das Verhindern vor unautorisiertem Zugriff auf das System.

3 Systemarchitektur

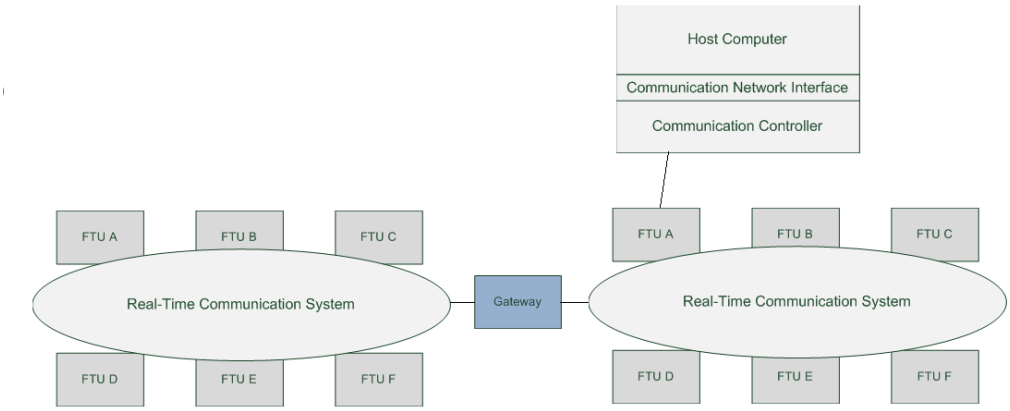

Abbildung 4: Architektur

Die Abbildung zeigt eine Struktur einer verteilten Echtzeitarchitektur. Auf der obersten Schicht besteht die Architektur aus einer Menge von Clustern, die über ein Gateway miteinander verbunden sind. Das Gateway dient dem Austausch von relevanten Informationen zwischen den Clustern. Ein Cluster besteht aus einer Menge von "Fehlertoleranten Einheiten"(Fault Tolerant Unit, FTU), die wiederum aus einer Menge von einem oder mehreren Knoten bestehen. Der Knoten besteht aus einer Menge von nebenläufig ausführbaren Tasks, die die gewünschte Funktion ausführen.
Mit Hilfe der Gateways kann diese Architektur beliebig erweitert werden, falls z.B. die Kapazität eines Clusters erreicht worden ist, wird einer der Knoten zu einem Gateway umfunktioniert.

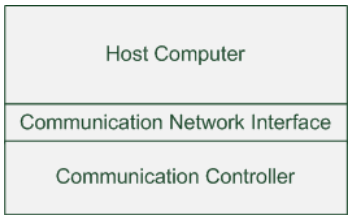

Abbildung 5: Ein Knoten

Ein Knoten ist eine Hard- und Softwareeinheit, welche sich in zwei Subsysteme(s. Abbildung 5) aufteilen lässt, nämlich in den lokalen Kommunikationscontroller (Communication Controller) und in den Host (Host Computer), welcher den Speicher und CPU enthält. Die Schnittstelle zwischen diesen beiden Systemen wird als Communication-Network-Interface (CNI) bezeichnet. Das CNI befindet sich auf der

Transportschicht des ISO-OSI-Referenzmodells und ist somit das wichtigste Interface eines verteilten Echtzeitsystems.

4 Echtzeitkommunikationssystem

4.1 Flusskontrolle

Flusskontrolle beschäftigt sich mit der Kontrolle der Geschwindigkeit des Informationsflusses zwischen Sender und Empfänger.
Flusskontrolle lässt sich in explizite und implizite Flusskontrolle aufteilen.
Bei der expliziten Flusskontrolle sendet der Empfänger explizit ein Acknowledgement (ACK) nach Erhalt der Nachricht zum Sender, um zu signalisieren, dass er die Nachricht erhalten hat. Im Kontrollbereich ist der Sender der Empfänger und somit für die Fehlerentdeckung zuständig.
Bei der impliziten Flusskontrolle wird voher festgelegt, wann Nachrichten gesendet werden. Um dies zu realisieren zu können, müssen die am Nachrichtenaustausch teilnehmenden Knoten über die gleiche Zeit verfügen, sie müssen also miteinander synchronisiert sein. Die Fehlerentdeckung liegt hier beim Empfänger.

4.2 Protokolle

Die Protokolle eines Kommunikationssystems lassen sich in Event-Triggered-Protokolle (ETP) und Time-Triggered-Protokolle (TTP) unterscheiden.

- **ETP**: Die Protokollausführung wird durch ein Event beim Sender zu irgendeinem Zeitpunkt ausgelöst. Für die Fehlerentdeckung ist der Sendeknoten zuständig, da nur dieser weiß, wann eine Nachricht gesendet wurde. Um so ein Protokoll realisieren zu können, wird explizite Flusskontrolle benötigt.
- **TTP**: Die Protokollausführung wird durch den Fortschritt der globalen Zeit ausgelöst. Die Fehlerentdeckung liegt beim Empfänger, da dies auf sein vorher festgelegtes Wissen basiert. Mit TTP kann eine zeitliche Kapselung der Knoten erreicht werden.

5 Fehlertolerante Systeme

In einem Echtzeitsystem können Fehler unterschiedlicher Art auftreten. Nachrichten können verloren gehen, Nachrichten können korrupt sein oder Knoten können ausfallen. Wichtig dabei ist, dass Hard- und Softwarefehler ein System nicht zum Totalausfall führen (Fehlertoleranz).
Folglich ist Fehlertoleranz besonders in sicherheitskritischen Systemen ein wichtiger Aspekt, da eben einzelne Komponentenfehler zu einer Katastrophe führen können. Um Fehler, welche eine Diskrepanz zwischen dem beabsichtigten Zustand und dem tatsächlichen Zustand des Systems darstellen, entdecken zu können, benötigt man Wissen über den beabsichtigten Zustand.
Die Fehlerentdeckung kann über festgelegte Constraints der Wissen über das korrekte Verhalten der Berechnung erlangt werden oder durch den Vergleich zweier Ergebnisse, die mit zwei redundanten Kanälen berechnet werden. Fehler müssen schnell angezeigt werden, was an der Schnittstelle zwischen Knoten und Kommunikationssystem geschieht. So kann gewährleistet werden, dass ein Knotenfehler schnell anderen Knoten im System mitgeteilt werden kann.
Es gibt sogenannte "Fehlertolerante Einheiten" (Fault Tolerant Units, FTUs), die Fehlertoleranz durch Replikation realisieren. Man erhält diese FTU mit Hilfe von "fail-silent"-Knoten, welche korrekte Ergebnisse produzieren oder keine. Der fail-silent-Knoten stellt sicher, dass intern Knotenfehler entdeckt werden. Wenn dieser implementiert wird, so kann die FTU mit zwei aktiven Knoten und einem Schattenknoten (shadow node) umgesetzt werden. In der folgenden Abbildung werden nur die beiden aktiven Knoten dargestellt.

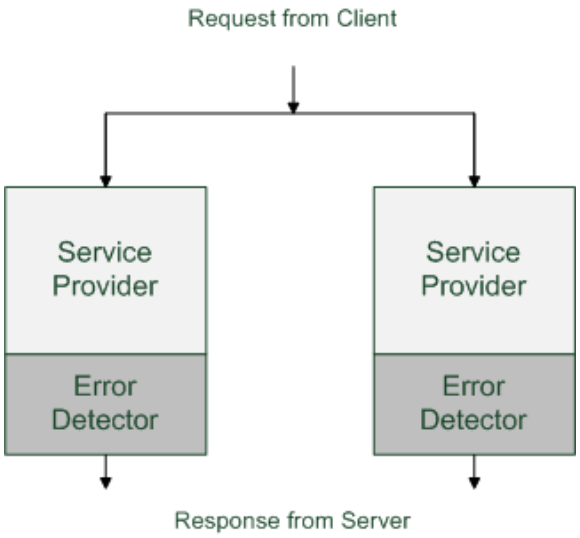

Abbildung 6: Fail Silent Knoten

Alle Knoten inklusive dem Schattenknoten lesen die Nachrichten vom Bus. Der Schattenknoten ist komplett mit den anderen Knoten synchronisiert, er produziert allerdings keine Ausgabe. Es werden also keine, eine oder zwei Nachrichten in diesem Beispiel erzeugt. Falls einer dieser aktiven Knoten ausfällt, tritt der Schattenknoten an dessen Stelle und wird zu einem aktiven Knoten. Ist der ausgefallene Knoten repariert worden, so wird er zu einem Schattenknoten.
Bei der Realisierung durch "Triple Modular Redundancy" sind alle beteiligten Knoten aktive Knoten, welche gemeinsam die Eingabe lesen und ein Ergebnis berechnen. Der Voter erhält dieses Ergebnis und ermittelt unter diesen Ergebnissen das Richtige, das dann als Antwort auf die Clientanfrage weitergeleitet wird.
Beispielsweise wird bei der Berechnung zweimal für das Bremsen gevotet und einmal für das Beschleunigen, so wird der Voter Bremsen für das richtige Ergebnis ansehen.

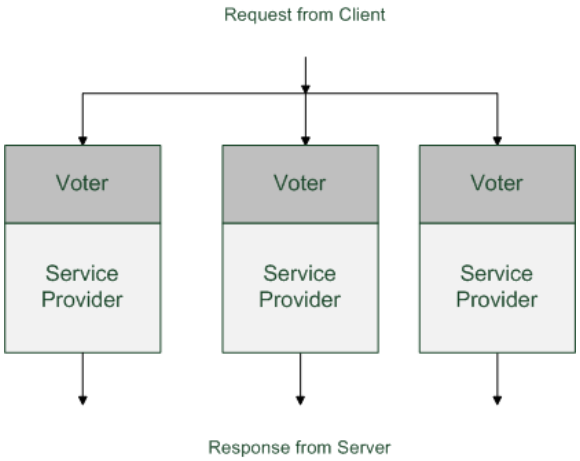

Abbildung 7: Triple Modular Redundancy

Literatur

[1] Kopetz, H. (2001) *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers

[2] Marwedel, P. (2003) *Embedded System Design*, Kluwer Academic Publishers