Implementing Cooperative Prefetching and Caching in a Globally-Managed Memory System

Daniel Hellmuth
Posener Weg 1
53119 Bonn

Fachbereich 2
7. Semester
WS 02/03

Inhaltsverzeichnis

1) Einleitung:

Dieser Vortrag geht über Prefetching und Caching in zukünftigen Systemen, die voraussichtlich global angelegt sind. Das heißt, CPU, Arbeitsspeicher und Festplatte werden nicht mehr als eine eigene Einheit gesehen, sondern als ein Verbund mehrerer als Ganzes. Diese Betrachtung ist dieselbe wie bei den verteilten Systemen.
Da im letzten Jahrzehnt, eine Vervielfachung der CPU-Geschwindigkeit von statten ging, aber in der Festplattenzugriffszeit kein so großer Fortschritt stattfand, versucht man an dieser Stelle anzusetzen und ein System zu finden, das dieses Nadelöhr vergrößert. Das Stichwort ist PGMS. Dabei handelt es sich um ein globales Prefetching- und Caching-System, das probeweise auf einem Unixsystem mit Myrinetverbindung implementiert wurde.
Dieses System mit seinem Algorithmus näher zu bringen wir die Aufgabe der folgenden Seiten sein.

Um die Festplattenzugriffszeit zu optimieren, hat man drei Ansätze miteinander verknüpft.

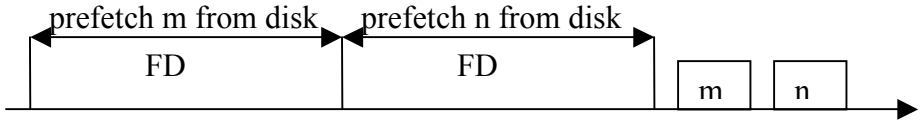
1) Prefetching von Festplattendaten in den Arbeitsspeicher anhand von Programmiererhinweisen oder Compileranmerkungen.
2) Das Verwenden von nicht benutztem Arbeitsspeicher eines Knotenpunktes als erweiterten Cachespeicher, da dieser globale Speicher viel schneller in einem Hochgeschwindigkeitsnetz wie dem Myrinet angesprochen werden kann.
3) Daten so auf verschiedene Speichermedien aufzuteilen um Sie parallel ansprechen zu können.

2) Definitionen:

Damit man dem Vortrag besser folgen kann, habe werden hier die wichtigsten Begriffe kurz zusammengefasst. Da es sich eigentlich um gängige Begriffe handelt, die schon bekannt sein sollten, wird nur das wichtigste in Gedächtnis zurückgerufen.

2.1 Konventionelles Prefetching (Gesteuertes Vorauslesen)

Aufgrund einer wahrscheinlichen Lokalität der Daten auf der Platte geht ein Programm davon aus, dass ein Zugriff auf den Block „x“einen Zugriff auf Block „x+1“zur Folge hat, auch wenn zum Zeitpunkt des Lesens noch kein expliziter Auftrag mit einem diesbezüglichem Parameter vorliegt. Aufgrund dieser Annahme liest eine Platte, einen gewissen Leerlauf vorausgesetzt, immer mehr Sektoren als angefordert in den internen Cache.
Wenn also Knotenpunkt „eins“ die Blöcke m und n in den Cache einlesen will, damit Sie zur rechten Zeit schnell verfügbar sind, kann das zu folgenden Problemen führen.
Zum einen kann der Speicher schon belegt sein, so dass das Einlesen nur mit einer gewissen Verzögerung stattfindet. Zum anderen kann das vorausschauende Einlesen von zukünftig benötigten Daten, Speicherstellen überschreiben die ebenfalls noch benötigt werden.



2.2 Konventionelles Caching

Ein Cache ist ein Zwischenspeicher zwischen langsamen und schnellen Einheiten, in dem häufig benötigte Daten abgelegt werden. Ein Cache wird zur Reduzierung von Zugriffszeiten benutzt. Caching befasst sich mit dem Algorithmus, welche Daten häufig benötigt werden und wie diese Daten in dem Cache gespeichert werden.

2.3 PGMS

PGMS (Prefetching Global Memory System) ist der Algorithmus für ein globales Speicher Management mit Prefetching. PGMS verknüpft, wie der Name schon vermuten lässt, drei bekannte Techniken in einem. Prefetching Algorithmen nach bekannten Verfahren von Patterson und anderen Autoren, GMS (Global Memory System) und den Verbund von Netzwerkknoten.

3) Globales Prefetching

In dem Falle eines globalen Speichers, hat ein Knotenpunkt drei Möglichkeiten für das Prefetching von Daten.

1) Von der Festplatte in den lokalen Speicher.
2) Von der Festplatte in den globalen Speicher. (In der Regel handelt es sich dabei um einen Speicher eines anderen Knotenpunktes.)
3) Von dem globalen Speicher in den lokalen Speicher.

Lokales Prefetching kann Probleme beinhalten, die wenigstens theoretisch durch globales Prefetching gelöst werden. Die Methoden dafür sind im Folgenden beschrieben.

a) frühzeitiges Ersetzen:
Das Problem das noch benötigte Daten vorzeitig ersetzt werden, wird bei globalem Prefetching dadurch gelöst, das ein Knotenpunkt „A“ einen anderen „B“ veranlasst die Daten in dessen Arbeitsspeicher vorzuhalten, bis Knotenpunkt „A“ sie benötigt. Da die Zeit um die Daten aus einem anderen Arbeitsspeicher zu lesen (wird hier als „Page fetch“ von globalem Speicher bezeichnet), bis zu 50-mal kleiner ist als das Lesen von Festplatte, ist immer noch ein enormer Zeitgewinn zu verbuchen. Diese Angabe bezieht sich natürlich nur Hochgeschwindigkeitsnetze, wie dem Myrinet mit 1Gb/sec.

b) I/O Bandweite
Die Bandweite bei einem Knotenpunkt ist auf die Schnittstelle zur Festplatte hin limitiert, in einem globalen Speicher, ist die Bandweite durch das parallele einlesen von Daten um ein vielfaches erweitert.

c) spekulatives Prefetching
Knotenpunkte innerhalb des Netzwerkes, die eine Zeit lang untätig sind, können das Seiten vorhalten, die vielleicht in naher Zukunft wieder benötigt werden.

4) Der globale Prefetching und Caching Algorithmus

Im folgenden Kapitel wird versucht den Aufbau des Algorithmus, vereinfacht darzustellen und anhand eines Beispiels zu verdeutlichen.
Da es sich um eine theoretische Betrachtungsweise handelt, müssen verschiedene Annahmen Zu Grunde gelegt werden. Das Laden einer Seite über das Netzwerk dauert immer die gleiche Zeit „Fg“ ebenso wie das Laden von Festplatte in den Arbeitsspeicher „Fd“. Es existiert schon ein zentraler Algorithmus, der die Programme aller Knoten und deren Referenzen auf Seiten festhält.

4.1 Caching Algorithmus
4.1.1 Aufgaben des Caching Algorithmus

- lokales Ersetzen von Seiten im Cache.
Die Bezeichnung ist etwas irreführend da in der Tat nicht der lokale Cache andere Werte oder Inhalte zugewiesen bekommt, vielmehr bedeutet das den Transfer von lokalen Seiten im Arbeitsspeicher in den globalen Speicher. Die Verarbeitungshierarchie bei lokalem Ersetzen von Seiten im Cache sieht vor, dass zuerst eine globale Seite vom lokalen Cache des Knotens an den globalen Speicher gesendet wird. Existiert im lokalen Cache keine globale Seite, wird diejenige lokale Seite in den globalen Speicher ausgelagert, deren Referenz vom Zeitpunkt der Entscheidung am weitesten in der Zukunft liegt.
- globales Ersetzen von Seiten im Cache.
Das Pendant zum lokalen Ersetzen. Es bedeutet das Herausschreiben von Seiten aus dem globalen Speicher. Damit ist noch nicht definiert, ob es sich um das Zurückschreiben auf Festplatte oder den Transfer in lokalen Speicher handelt. Für die Reihenfolge gilt: Es wird immer die Seite als nächstes aus dem globalen Speicher „entfernt“, deren Referenz am weitesten in der Zukunft liegt.

4.1.2 Beispiel für den Caching Algorithmus

- Eine Referenz von Knoten „A“ zeigt auf eine Seite „g“ im globalen Speicher auf Knoten „G“. „A“ lädt diese Seite „g“ in den lokalen Speicher, wo es zu einer lokalen Seite wird.
Im Austausch dazu wird eine Seite vom Knoten „A“ ausgelagert um wieder Platz zu schaffen. Wenn Knoten „A“ eine globale Seite im Speicher hat, wird diese Seite an den Speicher im Knoten „G“ gesendet. Sollte „A“ nur über lokale Seite und keine globalen Seiten verfügen, wird die lokale Seite deren Referenz am weitesten in der Zukunft liegt auf Knoten „G“ zu einer globalen Seite ausgelagert.
- Bei einer Referenz von Knoten „A“ auf eine Seite „d“ die sich auf der Festplatte befindet, wird die Seite „d“ in den Arbeitsspeicher von Knoten „A“ eingelesen. Um dafür im Arbeitsspeicher wieder Platz zu schaffen, ist diesmal ein Schritt mehr nötig. Zuerst wird wieder eine Seite „a“ aus dem lokalen Arbeitsspeicher in den globalen ausgelagert. Dazu wird dieselbe Methode wie oben beschrieben benutzt. Da ja nun eine Seite im globalen Arbeitsspeicher hinzugekommen ist, für die noch kein Platz geschaffen wurde, muss noch eine Seite „g“ aus dem globalen Arbeitsspeicher ausgelagert werden und zwar auf Festplatte. Dazu wird die Seite ausgelagert, deren Referenz am weitesten in der Zukunft liegt. Sollte sich diese Seite z.B. auf Knoten „G“ befinden, wird die Seite „g“ aus dem Speicher von „G“ auf Festplatte geschrieben und somit „a“ in den globalen Speicher.

4.2 Prefetching Algorithmus
4.2.1 Aufgaben des Prefetching Algorithmus

- lokales Prefetching.
Unter lokalem Prefetching versteht man das Einlesen von Festplatteninformationen in den lokalen Arbeitsspeicher oder den Transfer von globalem Arbeitsspeicher in den lokalen.
Das lokale Prefetching arbeitet nach dem Verfahren von Kimbrel, dieses Verfahren versucht die Seiten nur so zeitig zu laden das kein Zeitverlust bei der Verarbeitung von Programmabläufen auftritt.
- globales Prefetching.
Steht für das Einlesen von Festplatteninformationen in den globalen Arbeitsspeicherspeicher. Während das lokale Prefetching in dem PGMS nur zurückhaltend durchgeführt wird, geht der Algorithmus bei untätigen Knoten im globalen Netz sehr aggressiv vor. Für das Prefetching in den globalen Arbeitsspeicher wird der Ansatz von Cao verwendet. Wenn eine Seite von der Festplatte schneller zur Verfügung stehen muss als eine Seite im globalen Arbeitsspeicher, wird Sie auf jeden Fall zuerst in den globalen Arbeitsspeicher eingelesen. Um für diese Anforderung Platz im globalen Arbeitsspeicher zu schaffen, wird die Seite mit der „längsten“ Referenz im Netz, ausgelagert. Die Anzahl der lokalen Prefetches wird stark durch das aggressive globale Prefetching eingeschränkt.

4.2.2 Beispiel für lokales Prefetching

Betrachten wir den zu verarbeitenden Datenstrom an einem Knoten K zum Zeitpunkt T.
Annahmen:
m[x] = ist die x-te Seite, die nicht im lokalen Speicher des Knotens K zum Zeitpunkt T vorliegt und auch noch nicht zum bis zum Zeitpunkt T für den Prefetch vorgesehen ist.
Tm[x] = Zeit zwischen T und dem nächsten Zugriff auf m[x], vorausgesetzt es gibt kein Zeitverlust (stalling) zwischen T und diesem Zugriff.
Fd = Ist die Zeit die benötigt wird um eine Seite von der Festplatte des Knotens herunterzuladen.
Fg = Ist die Zeit die benötigt wird um eine Seite vom globalen Speicher, über das Netzwerk, zu laden.
Fi = Ist die Zeit die benötigt wird um m[x] in den lokalen Speicher zu laden.
Für „Fi“ gibt es zwei Möglichkeiten.
1) Fi liegt auf der Festplatte des Knotens => „Fi“ entspricht „Fd“
2) Fi liegt im globalen Speicher.=> „Fi“ entspricht „Fg“
Aufstellen der Formel um Zeitverlust zu vermeiden:
Prefetching ist nicht generell für jedes j nötig, nämlich wenn die Zeit die man benötigt um die ersten j-ten fehlenden Seiten zu erhalten ($\sum_{1 \le x \le j}$ Fi), kürzer ist als die Zeit die man bis zum Zugriff auf die j-ten fehlenden Seiten benötigt (Tm[j]).
Knoten K versucht also seine erste fehlende Seite m[j] herunterzuladen, wenn die Formel $\sum_{1 \le x \le j}$ Fi $\geq$ Tm[j] erfüllt ist.
Folgerung: Umso größer die Anzahl der fehlenden Seiten im globalen Speicher ist, desto später wird die Formel für lokales Prefetching auf True gesetzt.

5) Aufbau und Funktion von GMS

Mit GMS wurde versucht einen Teil dieses theoretischen Ansatzes, eines idealen Algorithmus, in einen Prototyp umzusetzen. Es handelt sich dabei um ein Unix basierendes globales Speichersystem.

GMS ist ein globales Speichersystem für ein Clusterverbund von Clients oder Workstations. Der Sinn besteht darin ein gemeinsames Speichersystem zu nutzen um die Durchschnittsgeschwindigkeit von allen Clusterapplikationen zu erhöhen.
Es sollen vor allem zwei Vorteile eines globalen Systems dabei zum Tragen kommen.
Bei einem Seitenaustausch kann die Seite um ein vielfaches schneller ausgelagert werden, wenn Sie in ein globales Speichersystem geschrieben wird, als auf Festplatte. Das gleiche gilt für ein erneutes Laden dieser ausgelagerten Seite.
Gemeinsame Seiten die von unterschiedlichen Knoten genutzt werden, können in einem globalen Speichersystem schneller ausgetauscht werden.
GMS setzt dabei auf einem bestehenden file-System und einem virtuellen Speichersystem, als eine Art von Betriebssystem auf.
Jede Seite in einem GMS, verfügt über eine Netzwerk weite einmalige ID.
(in der Regel entspricht Sie der Speicherstelle der Festplatte: IP- Adresse, device, inode, block offset). Damit alle Knoten eine UID auflösen können, verfügt ein GMS über ein Verzeichnis auf das alle Knoten zugreifen können und in dem alle Seiten im globalen Speichersystem referenziert sind. Diese Datenbank ist folgendermaßen aufgebaut.
PFD (per-node page-frame-directory). Enthält alle Einträge von Seiten zu einem bestimmten Knoten.
POD (page-ownership-directory). Weist eine UID einem übergeordneten Knoten zu. (wird hier als „manager node“ beschrieben) um die Informationen über das Verbleiben einer Seite zu erhalten, wenn Sie verschoben wird.
GCD (global-cache-directory). Ist eine verteilte Datenstruktur die einer eindeutigen ID eine IP-Adresse eines Knotens zuweißt.
Beispiel: Fragt ein Knoten eine bestimmte Seite an, findet dieser Knoten den „manager-node“ über die POD des gemeinsamen Verzeichnisse. Der „manager-node“ hält die Information vor, wo sich die Seite aufhält, und sendet eine Nachricht an diesen Knoten. Der Knoten wird vom „manager-node“ benachrichtigt, die angeforderte Seite an den Knoten mit der Seitenanforderung zu senden.

6) Aufbau und Funktion von PGMS

PGMS erweitert die Implementierung von GMS auf 3 Wegen.
- PGMS beinhaltet zusätzlich Operationen für das Prefetching von Blöcken in den Arbeitsspeicher eines Knotens vom globalen Arbeitsspeicher oder Festplatte.
- PGMS erweitert den Mechanismus von GMS um Informationen über eine globale Seite zu verwalten.
- PGMS berücksichtigt die Reihenfolge der Anforderung einzelner Applikationen und macht eine Aufstellung über die zeitliche Notwendigkeit einzelner Prefetches eines Verarbeitungsstreams.

Wir erinnern uns dass es auf der abstrakten Ebene zwei große Einteilungen für das Prefetching existierten. Zum einen das globale Prefetching (wird bei PGMS prefetch_to_global genannt) und das lokale Prefetching(prefetch_to_local).
Genau wie beim theoretischen Ansatz für lokales Prefetching wird beim prefetch_to_local wird eine Seite in den lokalen Speicher geladen. Befindet sich die Seite im globalen Speicher wird Sie über das Netzwerk geladen, sonst von der lokalen Festplatte.
prefetch_to_global lädt Seiten von der Festplatte des globalen Knotens in den globalen Speicher. Momentan werden im PGMS von den angeforderten Dateien Replikationen erstellt. Das heißt jeder Knoten der die Datei bearbeiten will erhält eine unabhängige Kopie. Damit keine Informationen verloren gehen, existiert ein gemeinsames Verzeichnis FAD (file-alias-directory) In diesem FAD wird für jede mehrfach verwendete Datei ein Eintrag gemacht, der die Ip-Addressen und den lokalen Filenamen für jede Kopie speichert.
Der FAD-Eintrag für einen File wird auf dem zuständigen „manager node“ gespeichert.
Durch die Erweiterung von PGMS durch FAD ist es möglich Dateien über mehrere Festplatten zu verteilen, im Gegensatz zu GMS. Seiten werden durch eine UID eindeutig bestimmt.

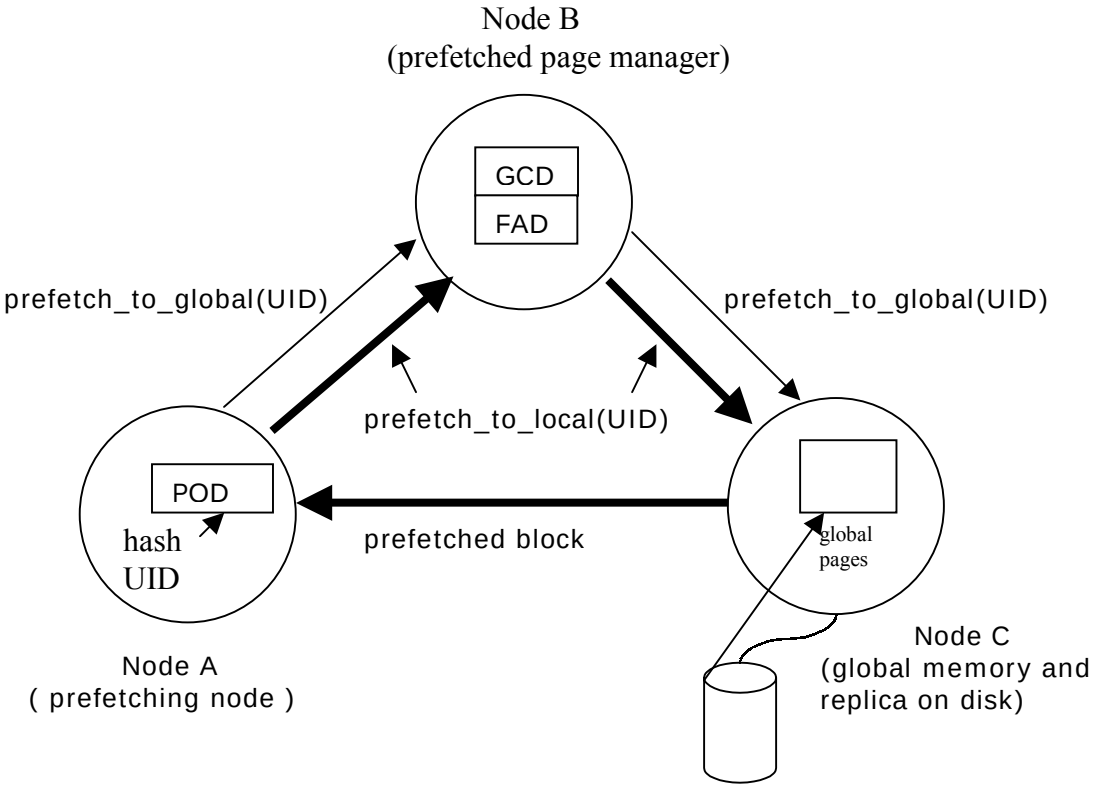
Beispiel:
Das Bild unten zeigt den Ablauf eines typischen prefetch_to_global. In diesem Fall handelt es sich um eine „shared page“. Gemeinsame Seiten oder Dateien werden im FAD jedes „manager nodes“ eingetragen.

→ (dünne Pfeile) = Zeigen die Schritte an die durchgeführt werden müssen wenn Knoten A eine Seite anfordert.

⇨ (dicke Pfeile) = Zeigen die Schritte an die später nach dem Prefetch vom globalen Speicher aus Knoten c in den lokalen Speicher von Knoten A nötig sind.

Die Anfrage wird zuerst vom Knoten A an den „manager node“ der entsprechenden Seite weitergeleitet. Dieser Knoten hält die Information ob und wo die Seite sich im Cache des Netzwerkes befindet. Wenn die Seite sich nicht im Cache befindet, wählt PGMS einen prefetching Knoten unter den unbenutzten Knoten aus, der dann die Datei der Seite repliziert. Wenn ein Knoten eine Anfrage auf eine Seite erhält, liest er die Seite von der Festplatte und cached Sie in den eigenen Speicher.

Abb. prefetch_to_global



7) Speedup (Performancesteigerung)

Technische Voraussetzung:
- 266 MHz DEC AlphaStation 500
- Digital Unix 4.0
- 1.28 Gb/s Myrinet
- 7200 RPM ST32171W Seagate Barricuda
- Seiten und Blockgröße = 8KB (Eine Zufallsseite von der Festplatte einzulesen benötigt 13ms).

Testlauf 1 - Microbenchmarks:
Aufbau:
Durchgeführte Microbenchmarks ergaben folgende Werte:
Diese Messungen beinhalten, dass es sich bei dem anfragenden Knoten um den gleichen Knoten handelt, wie dessen „manager node“. Für die meisten kleinen Netzwerke wird dies der Fall sein und wird deshalb als Normalfall angesehen. Die Spalte CPU gibt an welche Verarbeitungszeiten bei der CPU des jeweiligen Knotens anfallen und „Net“ entsprechend für die Netzwerkkommunikation.

1) prefetch_to_local Operation

Operation	Node		Time(µs)	
		CPU	Net	Total
Request	Requester	67.9	-	67.9
	Manager	53.9	35	88.9
Prefetch	Prefetcher	46.3	187(raw page transfer	233.3
Receive	Requester	22.3	-	22.3
Access	Requester	153	-	153

Erläuterung: Die Anfrage bei einem prefetch_to_local wird in 4 Komponenten oder Schritte aufgeteilt.
Request: Das Senden des Request für den Prefetch an den „manager node“ und die entsprechenden Zielknoten. Wie Abbildung „prefetch_to_global“ zeigt, wird erst die Anfrage an den „manager node“ geleitet und von da aus an den Zielknoten.
Prefetch: Aufbereitung der erhaltenen Nachricht und Senden der Seite an den anfragenden Knoten (Requester).
Receive: Der Requester erhält die Seite.
Access: Der Requester schreibt die Seite in die lokale „page map“.

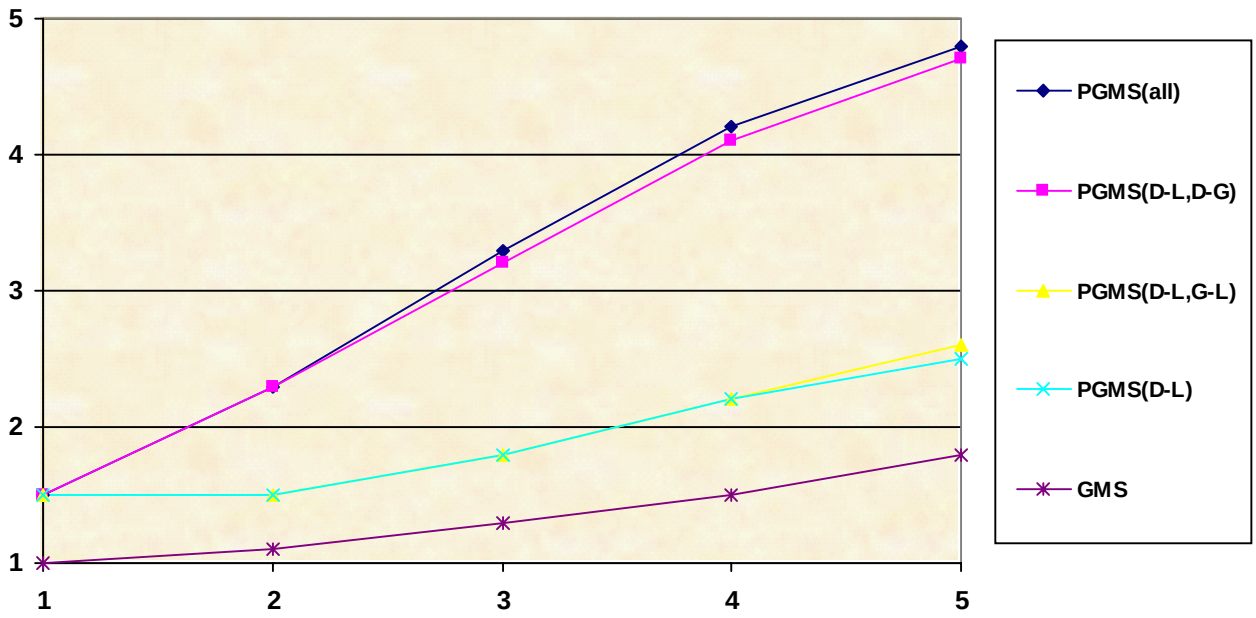
2) prefetch_to_global

Operation	Node		Time(µs)	
		CPU	Net	Total
Request	Requester	7.6	-	7.6
	Manager	40	35	75
Prefetch	Prefetcher	146	-	146

Der prefetch_to_global besteht aus 2 Schritten.

Request: Die Zeit um einen Request zu erstellen.
Prefetch: Die Zeit um eine Anfrage von der Festplatte in einen Arbeitsspeicher eines entfernten Knotens einzuleiten und anschließend auszuführen.

Testlauf 2 - Applikationstest:
Aufbau:
Es wurden mehrere Tests mit verschiedenen Applikationen durchgeführt um die Performancesteigerung durch den PGMS festzustellen. Alle Applikationen wurden auf einem Knoten mit 64 MB (davon 32 MB reservierter Speicher für Prefetching und Caching) durchgeführt. Zudem wurde eine variable Knotenzahl von bis zu 5 Knoten hinzugefügt, die über 32 MB globalen Speicher und eine einzelne Festplatte verfügten. Diese Knoten führten keine Programme aus und dienten lediglich als Arbeits- und Festplattenspeicher.
Ergebnis:
Die Auswertung der verschiedenen Tests kam zu dem Gesamtergebnis, das die Performancesteigerung für GMS mit jedem Knoten eine angemessene Steigerung brachte, wenn die Applikationen 3mal ausgeführt wurden. Während GMS eine Steigerung bei zyklischen Applikationstests brachte, waren die Ergebnisse bei PGMS sowohl für zyklische, als auch nichtzyklische Tests wesentlich besser. Es wurde für fast jede Applikation ein linearer Anstieg des Speedup mit der Anzahl der Knoten festgestellt.
Erläuterung:
Das bei GMS vor allem bei der zyklischen Durchführung ein Gewinn zu verbuchen ist, hängt damit zusammen, dass bei dem zyklischen Durchlauf die Seiten im globalen Speicher häufig wieder verwendet werden. Der Vorteil bei PGMS entsteht dadurch, das PGMS ein paralleles Prefetching ermöglicht.



Testlauf 3 – Abhängigkeit der Performancesteigerung zu den Prefetchmethoden.
Aufbau:
Anhand von einem Renderprogramm, wurde untersucht, welcher der verschiedenen Prefetchmethoden die beste Performancesteigerung ergibt. Diese Performancetests, wurden wieder auf einer variablen Anzahl von Knoten durchgeführt (1-5).
Ergebnis:
Als Ergebnis kam heraus, dass mit dem global_to_local prefetching, also dem Prefetching vom globalen in den lokalen Speicher des berechnenden Knotens kaum eine Steigerung erzielt wurde. Der größte Performancegewinn wurde durch den Schritt disk_to_global, dass Prefetching von Festplatte in den globalen Speicher, erzielt.
Erläuterung:
Global_to_local nutzt in diesem Fall nicht besonders viel, weil beim Rendern der Gewinn der durch das Cachen von ausgelagerten Seiten entsteht sehr gering ist. Ein zweiter Grund ist, das bei einem schnellen Netz wie in diesem Fall dem Myrinet nicht viel Zeitgewinn entsteht, wenn sich die Daten im lokalen Speicher aufhalten. Der hohe Gewinn bei disk_to_global, wird durch den Arbeitsspeicher der Knoten als einen Speicher für paralleles Prefetching erreicht.

Resultate der Performancemessungen:
Wenn Globales Prefetching und Caching als Erweiterung zum lokalen Prefetching eingesetzt wird, kommt es meist zu enormen Performancesteigerungen. Der Grund dafür ist die dazu gewonnene Parallelität bei Festplattenzugriffen, so das gesperrte Festplattenzugriffe in globale Speicherzugriffe umgewandelt werden.
Global_to_local Prefetches bringen bei schnellen Netzwerken kaum eine Performancesteigerung, weil der globale Zugriff sehr schnell ist.

8) Zusammenfassung

Es wurde die Herleitung eines Algorithmus für Prefetching und Caching in einem globalen Speichersystem erklärt. Dieser vorgestellte Algorithmus PGMS lädt die Seiten erst in den lokalen Speicher wenn es unbedingt nötig wird um Zeitverluste zu minimieren. Globales Prefetching wird hingegen aggressiv durchgeführt. Das Resultat dieses 2-wege Verfahrens ist, das der lokale Speicher möglichst lange erhalten wird und so die CPU die für den lokalen Speicher zuständig ist möglichst entlastet wird. Dieses Belassen des lokalen Speichers, wird dann wiederum durch aggressives globales Prefetching ausgeglichen. Der Vorteil dieses 2-wege Verfahrens ist, das die Gewichtsverlagerung der nötigen Prefetches nun von Knotenpunkten durchgeführt werden können, die zu einem Zeitpunkt nicht so stark oder gar nicht ausgelastet sind. Die Möglichkeit alte Seiten im Speicher auszutauschen ohne lokale CPU-Zeit zu benötigen ist der Hauptvorteil von Prefetching in einem globalen System.

9) Anhang

P. Cao, E. Felten, A. Karlin, and K. Li
A study of integrated prefetching and caching strategies

Implementing Cooperative Implementing Cooperative Prefetching and Caching in a Globally-Managed Memory System.
Department of Computer Science and Engineering University of Washington