DeFacto

Inhalt

Anlagen

DeFacto_Statistik (1 Seite)

Vorwort

Die Arbeiten begannen im Juni 2007. Damals begab ich mich zunächst allein auf die Suche nach einem neuen alternativen Faktorisierungsverfahren. Anlass hierfür war die unterrichts- und wettbewerbsbedingte Auseinandersetzung mit dem Verschlüsselungsverfahren RSA, deren Sicherheit durch das Faktorzerlegungsproblem begründet ist.

Nach einem Besuch der Seiten des RSA-Factoring-Challenges musste ich erstaunt feststellen, dass sich selbst hochrangige Mathematik- und Informatikfachleute in monate-, wenn nicht sogar jahrelangen Berechnungen verlieren.

Daraus erwuchs das Interesse, selbst ein Verfahren zu entwickeln. Nachdem ich mit dem ersten Konsolenentwurf meinen Informatiklehrer überraschte und keine ähnlichen Vorschläge im Internet finden konnte, war die Entscheidung für „Jugend Forscht“ gefällt.

Zunächst intuitiv, später dann theoretisch fundiert, kristallisierte sich die Überlegung einer Umsetzung im verteilten Netz heraus.

Das dies im Alleingang ziemlich schwer und unübersichtlich würde, schlug ich meinem informatikbegeisterten Mitschüler Christian Kohlert ein gemeinsames Projekt vor, ...

Burkhard Dietterle

... welches mittlererweile nicht mehr nur auf Einzelrechnern, sondern auf dem gesamten Schulnetzwerk aufbaut. Sogar ein eigener Projekt-Server wurde eingerichtet und befindet sich momentan in der Testphase. In naher Zukunft sollen über diesen RSA-Zahlen faktorisiert werden.

Unsere bisherigen Resultate erwecken die Hoffnung, dass sich unser System trotz anfänglicher Schwierigkeiten *defacto* mindestens ebenso gut verhält wie die momentan verwendeten.

Christian Kohlert

An dieser Stelle möchten wir uns herzlichst bei unserer Schule, dem Paulus-Praetorius-Gymnasium Bernau, für die großzügige Bereitstellung des Netzwerks bedanken. Nur so konnten wir unsere Entwürfe verwirklichen. Auch unserem Betreuer Hr. Neumeyer danken wir für das Mutmachen zur Teilnahme am Wettbewerb.

Bernau und Ahrensfelde, den 24. Januar 2008

Der DeFacto-Algorithmus

Einführung

Diese eher philosophisch als mathematisch gehaltene Einführung soll die Grundüberlegungen des DeFacto-Algorithmus unter Vorwegnahme der genauen Beschreibung darstellen.

Zuerst ist es notwendig, das allgemeine Problem der Faktorzerlegung auf ein Minimum zu reduzieren:

Gegeben sei eine ungerade natürliche Zahl p größer 3 mit der Aufgabe, eine Faktordarstellung $p = a \cdot b$ mit $1 < a, b < p$ zu finden oder die Primheit von p zu beweisen.

Das bezüglich der Erklärbarkeit einfachste Verfahren dieses Problem zu bewältigen, ist die wohlbekannte Probedivision. Ohne aber die Probedivision samt all ihrer Variationen im Detail auseinanderzunehmen, lässt sich das Verfahren abstrakt formulieren:

Man nehme eine Vermutung, eine Zahl a_V, berechne das Kriterium $p \bmod a_V$ und prüfe, ob das Kriterium gleich 0 ist. Ist dies der Fall, so ist mit a_V ein Teil der Lösung gefunden.

Zwei Tatsachen sind an dieser Stelle wertungsfrei festzustellen:

1. Der entsprechende Faktor b_E ergibt sich nicht unmittelbar aus der Vermutung oder dem Kriterium. Er entsteht erst durch $p \operatorname{div} a_V$, nachdem a_V als Faktor identifiziert wurde.
2. Für den Fall, dass das Kriterium ungleich 0 ist, lässt sich schlussfolgern, dass alle Vielfachen von a_V als weitere Vermutungen nicht mehr in Frage kommen. Um diesen Punkt zu nutzen bedarf es entweder einer speicheraufwendigen Primzahltabelle oder einer zeitaufwendigen Berechnung während des Vorgangs.

Die Übertragung der verwendeten Begriffe auf den DeFacto-Algorithmus lautet:

Die Vermutung ist ein Zahlenpaar (a_V, b_V). Das Kriterium ist $a_V \cdot b_V$. Ist es gleich p, ist die Lösung gefunden.

Wie später ausführlich gezeigt wird, ergibt sich im Vergleich mit der zweiten Feststellung zu der Probedivision Folgendes:

Das Ausschließen bestimmter Zahlen und Restklassen für weitere Vermutungen erfolgt nicht nach der Berechnung des jeweiligen Kriteriums, sondern noch während derselben. Die Kenntnis der Primzahlen spielt in DeFacto gar keine Rolle.

Der Ansatz ist demnach grundlegend anders:

Das Problem wird auf die Organisation und Multiplikation bestimmter Zahlenpaare hinausgeführt. Der Informationsgewinn eines Durchlaufes unterscheidet sich stark von dem der Probedivision.

Die umgekehrte Multiplikation

In diesem Abschnitt soll das Kernstück des DeFacto-Algorithmus erklärt werden, die umgekehrte Multiplikation.

Das Attribut „umgekehrt“ beschreibt eine wesentliche Eigenschaft des speziell für DeFacto entwickelten Multiplikationsverfahrens:

Es basiert auf der Kenntnis des Ergebnisses der Multiplikation.

Folgende Grafik zeigt die schriftliche Multiplikation von $45287 = 253 \cdot 179$ im dezimalen und binären Stellenwertsystem:

```
253x179          11111101x10110011
  253              11111101
 1771
 2277                  11111101
45287                   11111101

                              11111101
                               11111101
                      1011000011100111
```

Zunächst werden die Ziffern von b einzeln mit a multipliziert. Die Zwischenergebnisse werden in Blöcken versetzt untereinander geschrieben und addiert.

Betrachtet man nun p, a und b unter Zulassung vorangehender Nullen bei a und b als n-stellige Zahlen in einem Stellenwertsystem zur Basis s, so lässt sich die Multiplikation umkehren:

$$p = p_{n-1} \cdot s^{n-1} + \ldots + p_0 \cdot s^0$$
$$a = a_{n-1} \cdot s^{n-1} + \ldots + a_0 \cdot s^0$$
$$b = b_{n-1} \cdot s^{n-1} + \ldots + b_0 \cdot s^0$$

(1)	$p_0 = h_0 \bmod s$	$h_0 = a_0 \cdot b_0$	$h_0 \operatorname{div} s^n = 0$
(2)	$p_1 = h_1 \bmod s$	$h_1 = a_0 \cdot b_1 + a_1 \cdot b_0 + (h_0 \operatorname{div} s)$	$h_1 \operatorname{div} s^{n-1} = 0$
(3)	$p_2 = h_2 \bmod s$	$h_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0 + (h_1 \operatorname{div} s)$	$h_2 \operatorname{div} s^{n-2} = 0$
…	…	…	…
(n)	$p_{n-1} = h_{n-1} \bmod s$	$h_{n-1} = a_0 \cdot b_{n-1} + a_1 \cdot b_{n-2} + \ldots + a_{n-1} \cdot b_0 + (h_{n-2} \operatorname{div} s)$	$h_{n-1} \operatorname{div} s = 0$

Überträgt man nun die in der Einführung aufgestellte Aufgabe ins Binärsystem, bedeutet dies:

1. $s=2$
2. $n>2$
3. p_{n-1} bis p_0 sind gegeben
4. a_0 und b_0 sind jeweils gleich 1 (Faktoren eines ungeraden Produkts sind ungerade.).
5. a_{n-1} und b_{n-1} sind jeweils gleich 0 ($p \operatorname{div} x$ mit $x \geq 2$ ist im Binärsystem immer kürzer als p. (Trivialität)).

Aus der Tabelle ist erkennbar, dass von (i) zu $(i+1)$ jeweils nur zwei Unbekannte (a_i und b_i) hinzukommen. Die Vorgehensweise lautet deshalb:

1. Man beginne bei (2) .
2. Ist man bei (i) mit $1<i<n$, wähle man a_{i-1} und b_{i-1} so, dass (i) unter Berücksichtigung von $(i-1)$ erfüllt ist. Hat man eine Möglichkeit gefunden, gehe man zu $(i+1)$.
3. Hat man bei (i) keine (weitere) Möglichkeit, gehe man zurück zu $(i-1)$ und wähle eine alternative Kombination von a_{i-2} und b_{i-2} zur Erfüllung von $(i-1)$. Hat man eine solche Möglichkeit gefunden, gehe man wieder zu (i) .
4. Ist man bei (n) und ist (n) mit $a_{n-1}=b_{n-1}=0$ erfüllt, so ist die Lösung gefunden.
5. Kommt man bei (1) an, ist p prim.

Der DeFacto-Baum

In diesem Abschnitt soll der aus dem rekursiven Lösungsverfahren resultierende Entscheidungsbaum untersucht werden.

In jeden Schritt (i) lässt sich h'_{i-1} zunächst ohne $a_0 \cdot b_{i-1}$ und $a_{i-1} \cdot b_0$ berechnen:

$$h'_{i-1}=a_1 \cdot b_{i-2}+a_2 \cdot b_{i-3}+\ldots+a_{i-2} \cdot b_1+(h_{i-2} \operatorname{div} s)$$

Je nachdem, ob $h'_{i-1} \bmod s=0$ oder $h'_{i-1} \bmod s=1$ vorliegt, muss ($h_{i-1}=h'_{i-1}+0$ bzw. $h_{i-1}=h'_{i-1}+2$) oder $h_{i-1}=h'_{i-1}+1$ gerechnet werden, um $p_{i-1}=h_{i-1} \bmod s$ zu erfüllen.

Da a_0 und b_0 mit jeweils 1 vorgegeben sind, beschränken sich die Wahlmöglichkeiten für a_{i-1} und b_{i-1} auf ($a_{i-1}=b_{i-1}=0$ und $a_{i-1}=b_{i-1}=1$ (Typ A)) oder ($a_{i-1}=0 \wedge b_{i-1}=1$ und $a_{i-1}=1 \wedge b_{i-1}=0$ (Typ B)).

Es gibt also für jedes (i) mit $i \leq n-2$ maximal zwei Wahlmöglichkeiten, die es zu untersuchen gilt. Für $i=n-1$ sind a_i und b_i bereits vorgegeben. Dies entspricht dem Ausschließen von Restklassen für weitere Vermutungen, da bestimmte a_i und b_i gar nicht erst gewählt werden können.

Diese Wahlmöglichkeiten kann man in einem binären Entscheidungsbaum D organisieren:

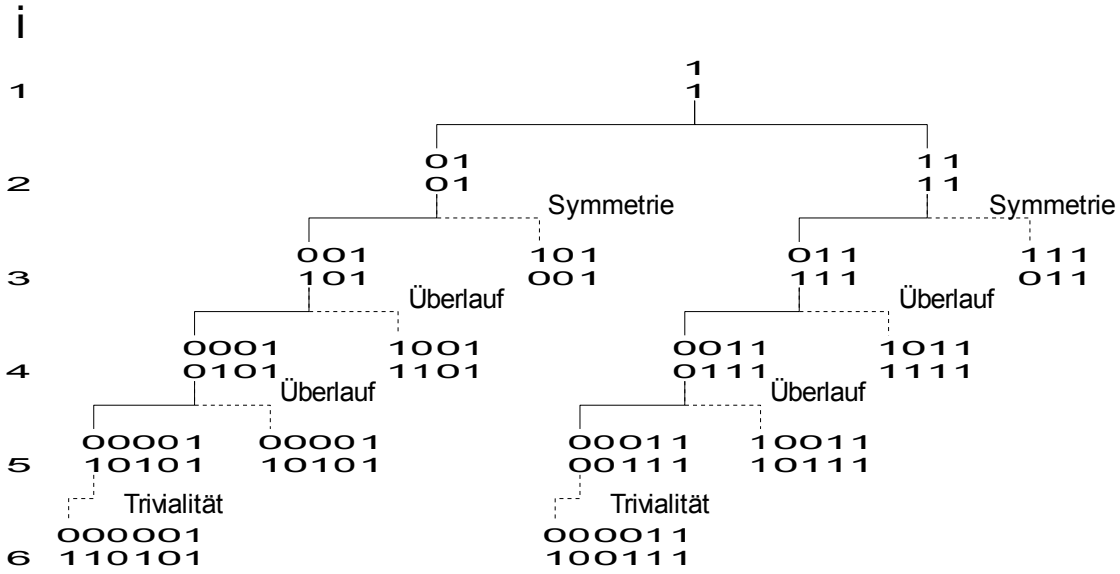
1. $(1),(2),\ldots,(i),\ldots,(n)$ entsprechen den Indizes der Ebenen von D .
2. Die Wahlmöglichkeiten entsprechen den Kanten von D .

Legt man folgende Ausrichtung für die Knoten der Ebene (i) von D fest, so lässt sich eine besondere Struktur erkennen:

$a_{i-1}=b_{i-1}=0$	Knoten links unten vom Vorgänger (Typ L)
$a_{i-1}=0 \wedge b_{i-1}=1$	
$a_{i-1}=b_{i-1}=1$	Knoten rechts unten vom Vorgänger (Typ R)
$a_{i-1}=1 \wedge b_{i-1}=0$	

1. Für alle Knoten der Ebene (i) , von denen Typ-B-Kanten ausgehen, entfällt die Typ-BR-Kante, wenn $a_{i-2}\cdot 2^{i-2}+\ldots+a_1\cdot 2+1=b_{i-2}\cdot 2^{i-2}+\ldots+b_1\cdot 2+1$, d.h. wenn der Knoten die Wurzel ist oder wenn der Pfad zu dem Knoten hin aus Typ-A-Kanten besteht. In diesem Fall würden, wenn überhaupt, der linke und rechte Teilbaum dieselben Lösungen enthalten (Symmetrie).
2. Für alle Knoten von (i) mit $i \geq n \operatorname{div} 2$ entfallen diejenigen Kanten, für die $h_i \operatorname{div} s^{n-i}=0$ nicht erfüllt werden kann. In diesem Fall würde $a_V \cdot b_V$ voraussehbar größer p (Überlauf).

Folgende Grafik zeigt den DeFacto-Baum für die Binärzahl 110101:



Erklärung: Oben stehen jeweils die bereits bekannten bzw. festgelegten Ziffern von a_V , unten die von b_V . Die gestrichelten Kanten fallen aufgrund der Trivialität, der Symmetrie oder des Überlaufs weg.

In der sechsten Ebene können bei dieser Zahl keine Knoten regelgerecht gebildet werden. Deshalb kommt man bei (1) an. Die Probe bestätigt die Behauptung: $110101_{[2]}=53_{[10]}$. 53 ist prim.

Dieser Beispielbaum zeigt bereits zwei wesentliche aufwandsoptimierende Eigenschaften der DeFacto-Bäume:

I. Jeder Pfad beschreibt ein ganz bestimmtes Zahlenpaar (a_V , b_V), von dem weder a_V noch b_V in irgendeiner weiteren Kombination auftreten.
II. Sie sind rechts von jedem Knoten überwiegend kürzer als links, da rechts die Typ-BR-Kante gegebenenfalls entfällt und die Häufigkeit der Überläufe aufgrund der Typ-R-Kanten größer ist als links (Dies entspricht ebenfalls dem Ausschließen bestimmter Restklassen für weitere Vermutungen).
Somit enthält ein DeFacto-Baum in der Realität weit weniger als 2^n-1 Knoten (im Beispiel 9 statt $2^6-1=63$).

Für jedes p gibt es einen eigenen eindeutigen DeFacto-Baum.

Suchmethoden in DeFacto-Bäumen

Dieser Abschnitt soll die Methoden zur Untersuchung von DeFacto-Bäumen beschreiben.

Wie in allen Binärbäumen bauen die zwei Strategien für die Lösungssuche auch in DeFacto-Bäumen auf den Grundkonzepten Tiefe-Zuerst- bzw. Breite-Zuerst-Suche auf.

Zunächst aber bedarf es einer weiteren Konvention:

Der zu untersuchende Bereich B von D sei das Intervall $[u,v]$ mit $u<v$, wobei u und v Indizes der (imaginären) Knoten auf der Ebene $(n-1)$ von D sind, d. h. selbst wenn diese Knoten nicht existieren. Die explizite Indizierung beginnt links bei 0 und endet rechts bei $2^{n-2}-1$ (Die relative Notation gibt die Lage relativ zur Wurzel an). Die Ebene (n) wird nicht berücksichtigt, da sie nur über Typ-AL-Kanten erreicht werden kann.

Die Bereichsgröße G von B ist gleich $v-u+1$.

Ein Bereich B mit $G>1$ ist in B_1 und B_2 mit $u_1=u, v_1=u_2-1$ und $v_2=v$ teilbar. Aufgrund der Eigenschaft I der DeFacto-Bäume lassen sich B_1 und B_2 parallelisieren.

Die Breite-Zuerst-Suche (Lösungssuche) in einem Bereich B entspricht dem systematischen Durchkämmen aller Knoten (inklusive der der Ebene (n)) in diesem Bereich. Nicht vorhandene Knoten werden gemäß Regelwerk durch die Rückkehr in die jeweils übergeordnete Ebene übersprungen.

Die in den vorangegangenen Abschnitten beschriebene Vorgehensweise entspricht somit der Breite-Zuerst-Suche im Bereich $[0, 2^{n-2}-1]$.

Aufgrund der Eigenschaft II der DeFacto-Bäume lässt sich aber auch die prinzipielle Möglichkeit des Vorhandenseins von Lösungen in einem bestimmten Bereich untersuchen, d. h. wenn es nicht möglich ist, regelgerecht einen äußeren Knoten der Ebene (i) mit $i \leq n-1$ zu bilden, so sind alle Knoten, die über diesen erreichbar wären imaginär und von vornherein aus dem zu untersuchenden Bereich ausschließbar. Es wird also ein B' mit $u' > u \wedge v' = v$ (Korrektur von links) oder $u' = u \wedge v' < v$ (Korrektur von rechts) gefunden.

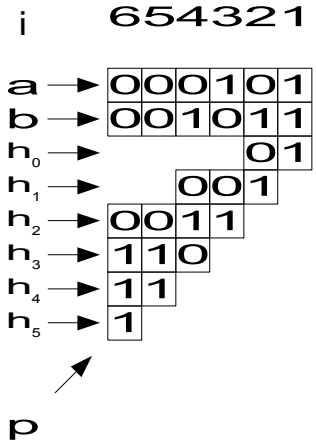
Die Tiefe-Zuerst-Suche bis zur Ebene (i) heißt Bereichskorrektur mit Genauigkeit $C = (i)$.

Die Bereichskorrektur liefert zwar keine Lösung, schränkt aber den Suchbereich stark ein. Sie ist dank geringerer Rekursionstiefe und regelbarer Genauigkeit ressourcenfreundlicher als die Lösungssuche.

Speicherverwaltung

Dieser Abschnitt soll einen kurzen Einblick in die Programmierung des DeFacto-Algorithmus geben. Wichtigstes Anliegen hierbei ist die Organisation der Daten in möglichst kompakter Größe und mit optimaler Zugriffszeit.

Folgende Grafik zeigt die Faktorisierungstabelle der umgekehrten Multiplikation für die Binärzahl 110111 beim Lösungsblatt (Bereichsindex 4):



Die Probe ergibt: $110111_{[2]} = 101_{[2]} \cdot 1011_{[2]} = 55_{[10]} = 5_{[10]} \cdot 11_{[10]}$.

Die Faktorisierungstabelle F besteht aus n Spalten. Die Spalten besitzen einen Kopf aus a_{i-1} und b_{i-1}. Weiterhin enthalten die Spalten einen Verweis auf p_{i-1} aus dem Hilfswertegitter H. Jedes p_{i-1} ist nach links hin mit h_{i-1} und nach oben hin mit h_{i-2} verknüpft.

Die Spalten sind untereinander nach links und rechts verknüpft.

Alle Ziffern werden als Wahrheitswerte dargestellt.

Diese Darstellung ermöglicht es mit minimalem Aufwand die umgekehrte Multiplikation durchzuführen:

Mit einem Hilfszeiger wird die Ebene (i) anhand der Spalte (i) festgehalten. Bei Rekursionsaufruf oder -rückkehr kann der Zeiger nach links bzw. nach rechts verschoben werden. Im DeFacto-Baum entspricht dies dem Wechsel auf eine tiefere bzw. höhere Ebene.

Durch Zugriff auf h_{i-2} lässt sich h'_{i-1}, h_{i-1} und p_{i-1} berechnen.

Während des Rechnens auf einer höheren Ebene bleiben alle Hilfswerte der tieferen Ebenen unverändert und sind bei Rekursionsrückkehr wieder abrufbar.

Die Operationen $\operatorname{div} 2$ und $\operatorname{mod} 2$ sind durch die gitterförmige Anordnung schon fest eingebaut. Die Zelle von p_{i-1} enthält $h_{i-1} \operatorname{mod} 2$. Die Nachbarzellen links neben p_{i-1} enthalten $h_{i-1} \operatorname{div} 2$. Die Zelle direkt über p_{i-1} und die Nachbarzellen links daneben enthalten $h_{i-2} \operatorname{div} 2$.

Überläufe erkennt man bereits vorzeitig an der nach links beschränkten Aufnahmefähigkeit des Hilfswertegitters ab $i = n \operatorname{div} 2 + 1$. Das stufenförmige Anwachsen des Hilfswertegitters für $i < n \operatorname{div} 2 + 1$ ist durch die Tatsache bedingt, dass in diesen Ebenen selbst bei maximalen a_V und b_V keine Überläufe entstehen können.

Die Abhängigkeit der Anzahl A der Zellen von H von n ist folgende:

rekursiv (aus der Tabelle ablesbar)	$A_1 = 1$ $A_i = A_{i-1} + i \operatorname{div} 2 + 1$
explizit (nach Umformung)	$A_i = i + \frac{1}{4} \cdot (i^2 - i \operatorname{mod} 2)$

Die Faktorisierungstabelle bleibt während des gesamten Berechnungsvorgangs unverändert. Der einzige dynamisch zu verwaltende Speicher ist somit der Stapel für die Rekursionsaufrufe.

Das Arbeiten mit Wahrheitswerten ermöglicht es, alle Manipulationen mittels Vergleichen, Verschieben und Überschreiben der Wahrheitswerte innerhalb der Faktorisierungstabelle zu lösen.

Das System ist plattform- und sprachenunabhängig.

Die Umsetzung innerhalb des DeFacto-Projektes erfolgte mit Borland Delphi 2005. Die DeFacto-EBA (Endbenutzerapplikation) ist auf der Projekt-Homepage herunterladbar. Sie dient insbesondere als Client-Anwendung für netzwerkbasierte Berechnung.

Die Umsetzung im Netzwerk

Einführung

Die Tatsache, dass Lösungen in beliebigen Bereichen des DeFacto-Baumes gesucht werden können, ohne dass dazu Kenntnis über das Ergebnis von Berechnungen in anderen Bereichen nötig wäre, ermöglicht die Aufteilung verschiedener Bereiche.

Die Aufteilung kann sowohl zeit-, als auch ortsunabhängig durchgeführt werden. Somit liegt die Verteilung des benötigten Rechenaufwands auf mehrere Computer nahe. Die Rechenzeit lässt sich im Vergleich zur Lösungssuche auf lediglich einem Computer drastisch verkürzen.

Die Koordination der Verteilung übernimmt die DeFacto-Verwaltung. Sie ist verantwortlich für die optimale Ausnutzung der zur Verfügung stehenden Rechner.

Die Aufgabe (p und der dazugehörige Bereich B) wird von einem Netzwerkauftraggeber formuliert, welcher auf jedem beliebigen Rechner im Netzwerk ausgeführt werden kann. Die Verwaltung führt anschließend die Bereichskorrektur durch um leere Teilbereiche des Baumes zu reduzieren.

Um die Leistung eines Computers für die Verwaltung zur Verfügung zu stellen, verbindet sich die DeFacto-EBA mittels TCP/IP und kann ab sofort Aufträge zur Lösungssuche in von der Verwaltung definierten Bereichen des Produkts empfangen.

Hat ein Rechner eine Lösung in dem ihm zugewiesenen Bereich gefunden, so sendet er diese an seine Verwaltung und diese wiederum an den ursprünglichen Netzwerkauftraggeber. Reste des Auftrages, welche möglicherweise noch auf anderen Rechnern bearbeitet werden, werden abgebrochen.

Sollte der zugewiesene Bereich keine Lösung erhalten, teilt die DeFacto-EBA dies ebenfalls seiner Verwaltung mit. Solange jedoch nicht der gesamte Baum auf Lösungen untersucht wurde erhält der Rechner sofort den nächsten Abschnitt. Wurde die vom Netzwerkauftraggeber übermittelte Zahl vollständig ohne Lösungsfund durchsucht, so wird diesem das Ergebnis (p ist prim) ebenfalls gesendet.

Alle für die Verwaltung nötigen Daten werden in zwei verschiedenen dynamischen Listen gespeichert.

Die Clientliste erhält zu jedem verbundenen Rechner einen Eintrag mit notwendigen Daten. Dazu zählen unter Anderem die Referenz auf das Socketobjekt, welches die Kommunikation über TCP/IP mit dem jeweiligen Rechner ermöglicht, die Adresse des Rechners im Netz, dessen Status, sowie weitere vom Clienttyp abhängige Daten.

In der Auftragsliste werden die von Netzwerkauftraggebern übermittelten Aufträge gespeichert, sowie die automatisch erstellen Teilaufträge, welche einen Teilbereich ihres Mutterauftrages beinhalten. Zusätzlich zu den essentiellen Auftragsdaten werden Bereichsgröße, Startzeit des Auftrages und für die Bereichskorrektur nötige Daten gespeichert.

Koordination der Ressourcen

Rechnern werden Aufträge zugewiesen, sobald diese sich als „wartend“ gemeldet haben und sobald offene Aufträge zur Verfügung stehen. Wird ein Teilauftrag einem Rechner zugeteilt, behält er ihn bis er ein Ergebnis geliefert hat oder sich von der Verwaltung getrennt hat. Im zweiten Fall wird der jeweilige Teilauftrag neu verteilt. Aus diesem Grunde dürfen die Rechenzeiten für jeden Teilauftrag nicht zu groß sein, da sonst beim Trennen der Verbindung die vom Rechner in einen Teilauftrag bereits investierte Rechenzeit verloren wäre.

Doch auch eine zu kurze Rechenzeit pro Teilauftrag wirkt sich negativ auf die Gesamtrechenzeit aus.

Für jede Verteilung eines Teilauftrages wird Prozessorzeit für die Teilung auf dem Server, sowie eine gewisse Zeit zum Versand über das jeweilige Netzwerk benötigt. Vor allem für die Kommunikation über das Internet kann pro Teilauftrag mit ca. 200ms gerechnet werden, was bei ständigem Verkehr einen großen Zeitverlust bedeuten kann.

Aus diesem Grunde wird eine Rechenzeit von 15 - 20 Sekunden von der Verwaltung angestrebt. Diese lässt sich erreichen, indem die Bereichsgrößen für jeden Rechner individuell dynamisch angepasst werden:

Benötigt ein Rechner für einen Auftrag zu lange, so wird der nächste Teilauftrag kleiner, schickt er das Ergebnis sehr schnell zurück, wird der nächste Bereich entsprechend größer.

Aufgrund der Eigenschaft II der DeFacto-Bäume, würde es in bestimmten Bereichen immer zu sehr kurzen (wenige Millisekunden langen) Rechenzeiten kommen. Die Verwaltung korrigiert leere Teilbereiche automatisch weg wenn sich diese sehr kurzen Berechnungen häufen.

Da die Bereichskorrektur mit zunehmender Genauigkeit sehr zeitintensiv werden kann, wird sie stets auf die unteren Ebenen des Baumes angewandt und nach mehrmaliger Wiederholung nur leicht erhöht.

Berechnung der optimalen Bereichsgröße

Die angestrebte Rechenzeit pro Teilauftrag von 15 – 20 Sekunden lässt sich nicht durch eine konstante Bereichsgröße erreichen, da die Bäume für größere p größere Leerbereiche besitzen, die sich in der Mitte von B befinden und somit durch die Bereichskorrektur nicht erfasst werden. Die der angestrebten Rechenzeit zugehörige optimale Bereichsgröße steigt mit wachsendem n an.

Bei der Erstellung des Auftrages in der Verwaltung wird einmalig eine anfängliche Bereichsgröße berechnet, welche bei ungefähr durchschnittlicher Rechenleistung (2,8Ghz Pentium4 Prozessor) die gewünschte Rechenzeit ergibt. Nach oben beschriebenen Verfahren wird diese in Laufzeit angepasst.

Die für die Berechnung verwendete Formel ließ sich durch Anpassen der Bereichsgrößen für Zahlen verschiedener Größenordnungen ermitteln. Da an einer genaueren Ermittlung der Formel zur Zeit noch gearbeitet wird, sollte die nun genannte Formel zunächst als Näherung angesehen werden:

$G_a = 2769100 \cdot e^{1,11030033 \cdot d}$

G_a : anfängliche Bereichsgröße

d : Anzahl der Dezimalstellen von p .

Interessant ist, dass G_a eine exponentielle Abhängigkeit (zur Basis e) von der Länge von p besitzt. Die genau Deutung dieser Beobachtung steht aber noch bevor.

Die Netzwerkstruktur

Hauptverwaltung

Um aufwendige Berechnungen z.B. zur Faktorisierung von RSA-Zahlen durchzuführen, muss die zuständige Verwaltung über einen langen Zeitraum ohne Unterbrechungen und Adresswechsel laufen. Dies ist auf den in unserem Heimgebrauch zur Verfügung stehenden Computern nicht möglich, vor allem wenn es sich um die Verteilung über das Internet handelt.

Aus diesem Grunde wird auf dem speziell zu diesem Zweck angemieteten vServer unter www.defacto-projekt.de eine angepasste Verwaltung ausgeführt, die DeFacto-Hauptverwaltung. Sie besitzt keine grafische Oberfläche wie die unter Delphi entwickelte Verwaltung, sondern ein Webinterface zur Anzeige der wichtigen Informationen. Des weiteren wurde sie für Linux portiert.

Aufträge werden wie gewöhnlich von einem Netzwerkauftraggeber gestellt, werden jedoch auch nach Trennung dessen Verbindung weiterbearbeitet um langwierige Berechnungen zu ermöglichen.

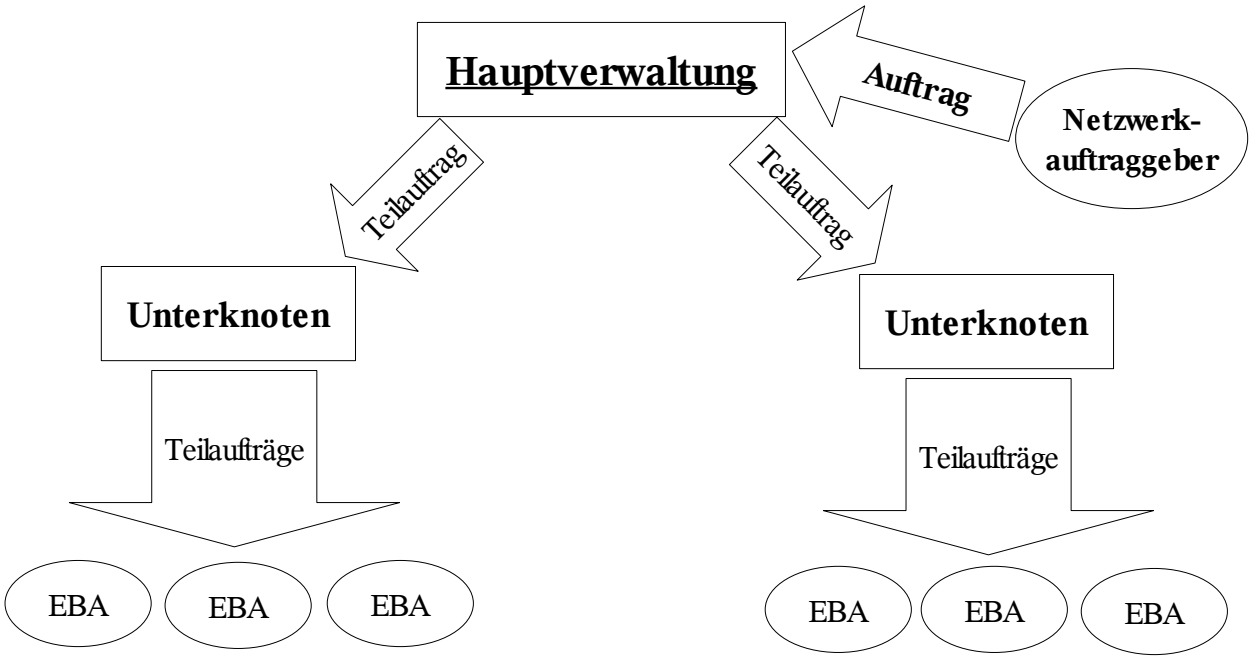
Unterknoten

Es können sich beliebig viele Rechner mit einer Verwaltung verbinden, solange diese fähig ist die nötigen Daten zu speichern und Anfragen zu bearbeiten. Sollte die Speicher-, Prozessor- oder Netzwerkauslastung zu hoch werden und Verzögerungen bei der Verteilung entstehen, ist der Einsatz von Unterknoten möglich.

Jede Verwaltung, außer die Hauptverwaltung selbst, kann sich als Unterknoten mit einer anderen Verwaltung verbinden.

Der Knoten dient dem Unterknoten als Auftraggeber und kann auf seine untergeordnete Rechnerkapazität zugreifen. Somit muss der Knoten lediglich so viele Rechner verwalten, wie er als Unterknoten besitzt, unabhängig von der Anzahl der ihm indirekt untergeordneten Rechnern.

Folgendes Diagramm veranschaulicht eine mögliche Netzwerkzusammenstellung:



Im obigen Beispiel existieren unter der Hauptverwaltung zwei Unterknoten, welche ihre Kapazität von je drei EBAs zur Verfügung stellen. In der Umsetzug wäre eine separate Verwaltung für nur drei EBAs logischerweise überflüssig. Dies dient hier lediglich der Veranschaulichung.

Neue EBAs, welche sich üblicherweise zuerst mit der Hauptverwaltung verbinden, werden so umgeleitet, dass sich die Kapazitäten der Unterknoten ausgleichen.

Ein Netzwerkauftraggeber formuliert einen Auftrag an die Hauptverwaltung, welche diesen in Teilaufträge für die Unterknoten aufteilt.

Jeder Unterknoten meldet die Anzahl der ihm untergeordneten Rechner seinem Knoten, damit dieser jedem Unterknoten eine seiner Kapazität entsprechende Auftragsgröße zuweisen kann. Dazu wird die Anzahl der Rechner eines Unterknotens mit der anfänglichen Bereichsgröße G_a von p multipliziert.

Jede EBA erhält nun einen Teilauftrag und sendet anschließend das Ergebnis an den Unterknoten. Wird der von der Hauptverwaltung übermittelte Bereich ohne Lösungsfund bearbeitet, fordert der Unterknoten einen neuen Teilauftrag an, ansonsten wird die Lösung an die Hauptverwaltung gesendet. Diese teilt das Ergebnis dem Netzwerkauftraggeber mit bzw. speichert es falls dieser nicht mehr verbunden ist.

Auswertung

Zunächst soll hier anhand der RSA-Zahl 140 (n = 140) der Speicheraufwand von DeFacto illustriert werden:

Den einzigen von n abhängigen Speicherbedarf erfordert die Faktorisierungstabelle. Diese besteht für $n = 140$ aus 140 Spalten à 14 Byte und A_{140} = 5040 Zellen à 9 Byte. Insgesamt also 47320 Byte (46,2 KB). Selbst bei großzügigem Überschlag von 5 MB für die obligatorischen Programmdaten erfordert DeFacto pro Rechner weitaus weniger Arbeitsspeicher als die entsprechende Lösung innerhalb des RSA-Factoring-Challenges (26 Mbyte/Rechner bei Matrizen von 4671181×4704451).

Nun eine Einschätzung des Zeitaufwandes (siehe Tabelle im Anhang):

Der Zeitaufwand ist nicht direkt von der Größe, auch nicht der Länge von p abhängig.

Der Zeitaufwand ist abhängig von der Anzahl der tatsächlichen Teiler: DeFacto ist tendenziell desto schneller, je mehr Teiler p besitzt. (Allerdings entfällt für uns dieser Bewertungspunkt, da bei RSA nur Semiprimzahlen verwendet werden.)

Das eventuell als Manko bewertete späte Auffinden „leichter“ Teiler (3, 5, 7, 11 u.s.w) wird für uns durch die Tatsache ausgeglichen, das große Teiler vergleichsweise schnell gefunden werden. Dies liegt daran, das Aufgrund der umgekehrten Multiplikation die Größe von a_v und b_v und deren Differenz von einem Knoten zum anderen sprunghaft größer bzw. kleiner wird. Dies lässt vermuten, dass die Kreierung einer „besonders“ schweren Semiprimzahl für DeFacto unmöglich ist.

Durch die einfache Umsetzung und den Ressourcenausgleich im Netzwerk sind keine Supercomputer nötig. Unser Ziel ist es, über das Internet ein Netzwerk aus freiwilligen Teilnehmern, die jeweils ihre Rechnerleistungen beisteuern, aufzubauen, um somit RSA-Zahlen zu zerlegen.

Quellenangabe

alle Seiten zuletzt abgerufen am: 24. Januar 2008

http://de.wikipedia.org/wiki/Faktorisierungsverfahren, Wikipedia, Faktorisierungsverfahren

http://en.wikipedia.org/wiki/Integer_factorization, Wikipedia, Faktorisierung von Ganzzahlen

http://www.rsa.com/rsalabs/node.asp?id=2092, RSA Security, RSA Factoring Challenge