Der Algorithmus TUPAL für Warteschlangennetze mit Produktform

The algorithm TUPAL for queueing networks with product form

I. F. Akyildiz, Universität Erlangen-Nürnberg

In dieser Arbeit wird ein Algorithmus für Produktformnetze vorgestellt. Der Algorithmus basiert auf der Durchsatzformel, über die dann durch Anwendung der entsprechenden Formeln die anderen Leistungsgrößen bestimmt werden können. Der Algorithmus ist zur Unterstützung der Anwendung von Nortons Theorem auf Warteschlangennetze bzw. der Methode von Marie sehr geeignet.

In this work an algorithm is presented for product form queueing networks. The algorithm is based on the throughput formula which is used to determine other performance measures by applying the according formulae. The algorithm can also be used for the application of the parametric analysis and the method of Marie.

1 Einführung

Die Leistungsanalyse von Rechensystemen erfolgt im wesentlichen aus drei Gründen:

- aus einer Anzahl bereits bestehender Rechensysteme sollen diejenigen ausgesucht werden, die für die eigenen Zwecke am geeignetsten sind;
- die Leistungsfähigkeit eines noch nicht bestehenden Systems soll vorhergesagt werden, um die Effizienz neuer Konzepte und Strukturen des Entwurfs zu untersuchen;
- man versucht, Engpässe in einem Rechensystem zu bestimmen.

Die Leistungsanalyse von Rechensystemen wird durch Meßmethoden oder Modellierungstechniken durchgeführt.

Bei existierenden Systemen kann die Leistung direkt durch Meßmonitore (Hard-/Software-Monitore) ermittelt werden. Obwohl die Messung am realen System ein wichtiger Bestandteil jeder Leistungsanalyse ist, stehen einer ausschließlichen Verwendung von Meßtechniken Probleme entgegen; wie z. B., daß während der Entwurfs- und Entwicklungsphase eines Systems Messungen noch nicht durchführbar sind.

Da zudem das Ablaufgeschehen in heutigen Rechensystemen häufig zu komplex ist, um nur durch Messungen erfaßt oder vorhergesagt zu werden, werden zur Analyse Modelle verwendet. Anhand dieser Modelle kann das dynamische Ablaufgeschehen in Rechensystemen bewertet und optimiert werden. Damit können reale Abläufe formal beschrieben, charakteristische Leistungsgrößen wie z. B. Auslastung der Geräte, Durchsatz des Systems, Verweil- und Wartezeiten der Prozesse in den einzelnen Geräten usw. definiert und bestimmt und Entscheidungshilfen für den Entwurf optimaler Hardwarestruktur und Betriebsprogramme bereitgestellt werden.

Eine Klasse von Modellen, die sich besonders für solche Zwecke eignen und seit 15 Jahren ein großes Interesse gefunden haben, sind die Warteschlangennetzmodelle (queueing network models).

Ein Rechensystem, in dem sich Aufträge (jobs) zwischen den Bedienstationen (Geräte + Warteschlangen) bewegen, um ihre Bedienanforderungen zu erfüllen, kann durch ein Warteschlangennetz modelliert werden.

Man unterscheidet zwischen offenen, geschlossenen und gemischten Netzen. Ein Warteschlangennetz ist *offen*, wenn Aufträge von außen und Abgänge nach außen von jeder Bedienstation möglich sind. Offene Netze sind geeignet, um Rechnernetze zu modellieren.

Ein Warteschlangennetz ist *geschlossen*, wenn die Anzahl der Aufträge im Netz immer konstant ist, d. h., sobald ein Auftrag das System verlassen hat, rückt sofort ein neuer nach, so daß die Anzahl der Aufträge im System immer konstant bleibt. Geschlossene Netze werden häufiger angewandt, da sie das Verhalten vieler Rechensysteme besser modellieren.

Ein Warteschlangennetz ist *gemischt*, wenn es aus offenen und geschlossenen Teilketten besteht. Ein Rechensystem für Online-Betrieb und Batch-Betrieb kann z. B. als ein gemischtes Netz modelliert werden. Der Online-Betrieb wird durch die geschlossene Kette (die Anzahl der Benutzer konstant), der Batch-Betrieb durch die offene Kette dargestellt.

Warteschlangennetze können auf verschiedene Weise untersucht werden:

i) *Analytische Methoden* leiten auf mathematischem Wege die Beziehungen zwischen fundamentalen Systemparametern, z. B. Bedienzeiten, Anzahl der Aufträge, Übergangswahrscheinlichkeiten, und relevanten Leistungsgrößen, z. B. Auslastung, Durchsatz, Warteschlangenlänge, Wartezeit, Verweilzeit usw., her.

ii) Bei der *Simulation* werden die Vorgänge in Rechensystemen mit speziellen Programmen nachgespielt, die in üblichen Programmiersprachen oder mit eigens dafür entwickelten speziellen Simulationssprachen formuliert werden.

iii) *Hybrid-Simulation* ist eine Kombination der analytischen Modellbildung mit der Simulation. Bei der Simulation eines umfangreichen Systems können für Teilsysteme analytische Modelle gegeben sein, deren Simulation entfallen kann oder umgekehrt.

Wegen der einfachen Implementierung und der leichten Interpretation der Beziehungen zwischen Modellparameter und Leistungsgrößen und der damit verbundenen leichten Optimierung und der Entwicklung neuer Methoden haben die analytischen Methoden in den letzten Jahren zunehmend an Bedeutung gewonnen.

In dieser Arbeit wird eine neue analytische Methode TUPAL (Throughput Algorithm) eingeführt, die auf Produktformnetze anwendbar ist. Zunächst werden im nächsten Abschnitt die Produktformnetze erläutert.

2 Produktformnetze

Ein Netzwerk hat *Produktformlösung*, wenn jede Bedienstation von einem der folgenden Typen ist:

Typ 1 M/M/m – FCFS
Typ 2 M/G/1 – RR-PS
Typ 3 M/G/IS – .
Typ 4 M/G/1 – LCFS-PR

[1; 2; 3] haben gezeigt, daß sich die Wahrscheinlichkeit für einen gegebenen Systemzustand durch das Produkt der Zustandswahrscheinlichkeiten (Randwahrscheinlichkeiten) der einzelnen Stationen ergibt:

$$p(k_1, \ldots, k_N) = \frac{1}{G(K)} \prod_{i=1}^{N} p_i(k_i), \qquad (1)$$

wobei

N Gesamtzahl der Stationen.
K Gesamtzahl der Aufträge.
k_i die Anzahl der Aufträge in der i-ten Station ist.
G(K) ist die Normalisierungskonstante, die die Summe aller Wahrscheinlichkeiten auf 1 normiert. (Für offene Netze gilt G = 1.)
$p_i(k_i)$ sind die Randwahrscheinlichkeiten, die anhand der mittleren Bedienzeiten $1/\mu_i$ und der relativen Besuchshäufigkeiten e_i der Aufträge zu der i-ten Station berechnet werden können.

Die relativen Besuchshäufigkeiten lassen sich aus dem folgenden Gleichungssystem ermitteln:

$$e_i = \sum_{j=1}^{N} e_j p_{ji}, \qquad (2)$$

wobei p_{ji} Die Übergangswahrscheinlichkeit von der j-ten zur i-ten Station ist.

Da es im geschlossenen Netz (N – 1) unabhängige Gleichungen gibt, nimmt man $e_1 = 1$ an.

Verschiedene Bedingungen müssen erfüllt sein, damit ein Warteschlangennetz Produktformlösung hat. Diese Bedingungen fassen wir wie folgt zusammen:

a) M → M-Eigenschaft [4]
Eine Station hat die M → M-Eigenschaft genau dann, wenn sie einen Poissonschen Ankunftsprozeß in einen Poissonschen Abgangsprozeß transformiert.

b) „LOCAL BALANCE"-Eigenschaft [5]
Ein Netzwerk ist im lokalen Gleichgewicht genau dann, wenn die Rate, mit der ein Zustand aufgrund des Abgangs eines Auftrags aus einer Station verlassen wird, gleich derjenigen Rate ist, mit der dieser Zustand aufgrund des Übergangs eines Auftrags in diese Station wieder erreicht werden kann.

c) „STATION BALANCE"-Eigenschaft [6]
Eine Warteschlangenstrategie erfüllt die Station-Balance-Eigenschaft, wenn die Rate, mit der Aufträge in einer Position der Warteschlange bedient werden, proportional zur Wahrscheinlichkeit ist, daß ein Auftrag diese Position betritt. Mit anderen Worten wird die Warteschlange einer Station in einzelne Positionen unterteilt, und die Raten, mit denen diese Positionen betreten bzw. verlassen werden, werden gleichgesetzt.

In [6] wird gezeigt, daß ein Netz, das die Station-Balance-Eigenschaft erfüllt, auch die Local-Balance- und die M → M-Eigenschaften erfüllt und Produktformlösung hat. In der gleichen Arbeit wird auch gezeigt, daß Typ-2,3,4-Stationen die Station-Balance-Eigenschaft erfüllen und Typ-1-Stationen nicht. Daher haben Stationen mit FCFS-Strategie und nichtexponentieller Bedienzeitverteilung keine Produktformlösungen. Die Beziehungen der Eigenschaften werden in Bild 1 deutlich.

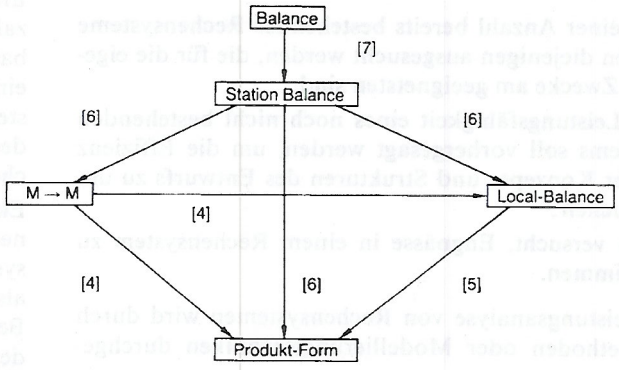

Bild 1. Beziehungen der Eigenschaften.

● *Erweiterungen*

– *Typ-5-Station* M/M/1-SIRO
[8] analysiert die Strategie SIRO (Service in Random Order) und zeigt, daß diese Stationstypen wie Typ 1 Produktformlösungen haben.

– *Typ-6-Station* M/M/1-LBPS
[9] führt die Strategie LBPS (Last Batch Processor Sharing) ein und zeigt, daß die Lösungen für solche Stationstypen Produktform haben. Bei dieser Strategie wird der Prozessor zwischen den letzten Batch-Aufträgen verteilt. Falls der letzte Batch einen einzigen Auftrag enthält, hat man LCFS-PR; falls er alle Aufträge enthält, hat man PS.

– *Typ-7-Station* M/M/1-WEIRDP (und BALANCE-Konzept)

[7] behandelt ein neues Konzept „BALANCE", mit dem er beweist, daß Preemptive Prioritätsstrategien nicht das Balance-Konzept erfüllen und daher keine Produktformlösung haben können, es sei denn, daß alle Aufträge eine exponentielle Bedienzeitverteilung mit gleichen Mittelwerten besitzen. Das Balance-Konzept bezieht sich auf die Warteschlangenstrategie und existiert dann, wenn eine Strategie eine charakteristische Funktion besitzt, die alle nicht-negativen Integervektoren k (Zustände) als Definitionsbereich hat. Für die Bestimmung der charakteristischen Funktion wird ein iterativer Algorithmus gegeben und mit Hilfe eines Theorems gezeigt, daß die Balance-Eigenschaft in der Station-Balance enthalten ist. Auf dem Balance-Konzept basierend behandeln [7] die Strategie WEIRDP. Bei dieser Strategie wird dem ersten Auftrag in der Warteschlange ein Teil p des Prozessors und $(1-p)$ den restlichen Aufträgen zugeteilt. Sie geben eine Produktformlösung für solche Stationstypen und zeigen, daß das Konzept von [9] richtig ist.

3 TUPAL (Throughput-Algorithm)

Bei der Analyse von Warteschlangennetzen wurden früher alle Leistungsgrößen eines Systems mit Hilfe der Zustandswahrscheinlichkeiten (Gl. 1) berechnet. Um die Zustandswahrscheinlichkeiten zu berechnen, muß, wie aus Gl. (1) ersichtlich, die Normalisierungskonstante im Falle geschlossener Netze ermittelt werden.

Ein umständlicher Weg für die Berechnung von G(K) ist die Aufzählung aller Zustände und die Bestimmung deren relativer Wahrscheinlichkeiten. Absolute Wahrscheinlichkeiten können dann aus den relativen Wahrscheinlichkeiten durch Normierung von deren Summe zu 1 bestimmt werden. Dies ist natürlich nur für kleine Netze durchführbar, weil für größere Netze die Anzahl der Zustandswahrscheinlichkeiten auch dementsprechend zunimmt.

[10] hat einen effizienten Algorithmus für G(K) entwikkelt, ohne die Zustände des Netzes in die Berechnung einzubeziehen. Mit diesem Algorithmus ist es möglich, die Leistungsgrößen mit einfachen Formeln zu ermitteln. Inzwischen gibt es mehrere, zum Teil sehr unterschiedliche Algorithmen zur Bestimmung der Normalisierungskonstanten. Einer der Algorithmen, die große Anerkennung gefunden haben, ist der Faltungsalgorithmus (=Convolution Algorithm) von [10; 11], der von [12] auf Mehrklassennetze erweitert wurde. Der andere populäre einfache Algorithmus ist der von der MVA abgeleitete LBANC von [13].

Da die Normalisierungskonstante Over-/Underflow-Probleme mit sich bringt, hat man versucht, die Leistungsgrößen ohne Zuhilfenahme der Normalisierungskonstanten zu berechnen. Dabei entstand die MVA [14], die in den letzten Jahren große Anerkennung gefunden hat. Mit der MVA lassen sich iterativ unter Verwendung von drei fundamentalen Gleichungen für die mittlere Verweilzeit, den Durchsatz und die mittlere Anzahl von Aufträgen die Leistungsgrößen bestimmen. Sie ist leicht implementierbar und ist sehr schnell. Für Netze mit lastabhängigen Stationen ($m_i > 1$) und mehreren Auftragsklassen hat sie jedoch Nachteile, wie z. B. Speicherplatz-, Stabilitäts- und Rundungsfehlerprobleme.

Im folgenden Algorithmus soll gezeigt werden, daß die Iteration für die Bestimmung der Leistungsgrößen nur über eine einzige Formel (Durchsatz-Formel) durchgeführt und daraus dann alle anderen interessierenden Leistungsgrößen bestimmt werden können.

● *Theorem*

In einem geschlossenen Warteschlangennetz mit Typ-1,2,3,4,5,6,7-Stationen ergibt sich der Durchsatz des Systems aus der folgenden Formel:

$$\lambda(K) = \begin{cases} K \cdot \left[\sum_{l=1}^{K} \prod_{j=1}^{l-1} \lambda(K-j) \sum_{i=1}^{N} x_i^l \right]^{-1} & \text{Typ 1, 2, 3, 4, 5, 6, 7} \\ K \cdot \left[\prod_{j=1}^{K-1} \lambda(j) \sum_{i=1}^{N} \sum_{l=1}^{K} l \cdot p_i(l/K) \right]^{-1} & \text{für Typ 1 } (m_i \geq 1) \\ K \cdot \left[\sum_{i=1}^{N} x_i \right]^{-1} & \text{Typ 3} \end{cases} \quad (3)$$

wobei $x_i = \frac{e_i}{\mu_i}$ die relative Auslastung der i-ten Station ist. e_i ermittelt man aus Gl. (2). (4)

$p_i(l/K)$ ist die Wahrscheinlichkeit, daß in der i-ten Station l Aufträge bedient werden, unter der Bedingung, daß sich im Gesamtsystem K Aufträge befinden. Sie ergibt sich aus:

$$p_i(l/K) = \frac{x_i \cdot p_i(l-1/K-1)}{\gamma_i(l)} \quad (5)$$

mit

$$p_i(O/K) = \frac{1}{\prod_{j=1}^{K} \lambda(j)} - \sum_{l=1}^{K} p_i(l/K) \quad (6)$$

$$p_i(O/O) = 1 \qquad p_i(l/O) = O \qquad \forall i = 1, \ldots, N$$

Die Funktion $\gamma_i(l)$ berechnet man aus der Formel:

$$\gamma_i(l) = \begin{cases} l & l \leq m_i \\ \text{für} & \\ m_i & l \geq m_i \end{cases} \quad (7)$$

Als Initialwert nimmt man

$$\lambda(1) = \frac{1}{\sum_{i=1}^{N} x_i}$$

Der Algorithmus bricht dann ab, wenn die Gesamtzahl der Aufträge K erreicht ist.

● *Beweis*

i) für $m_i = 1$:
Man geht aus von der Formel für den Durchsatz,

$$\lambda(K) = \frac{G(K-1)}{G(K)}, \tag{8}$$

für die Normalisierungskonstante

$$G(K) = \frac{1}{\prod_{j=1}^{K} \lambda(j)} \tag{9}$$

und

$$G(K) = \frac{1}{K} \sum_{i=1}^{N} \sum_{l=1}^{K} x_i^l G(K-l). \tag{10}$$

Die Gleichung (10) kann aus der Formel für die mittlere Anzahl von Aufträgen in der i-ten Station abgeleitet werden [15]:

$$\bar{k}_i(K) = \sum_{l=1}^{K} x_i^l \frac{G(K-l)}{G(K)}$$

Beide Seiten der Gleichung werden von $i = 1, \ldots, N$ summiert:

$$\sum_{i=1}^{N} \bar{k}_i(K) = \sum_{i=1}^{N} \sum_{l=1}^{K} x_i^l \frac{G(K-l)}{G(K)}$$

Da die Anzahl der Aufträge in einem geschlossenen Netz konstant ist, erhält man

$$K = \sum_{i=1}^{N} \sum_{l=1}^{K} x_i^l \frac{G(K-l)}{G(K)}.$$

Nach Umformung nach G (K) erhalten wir Gl. (10). Nun wieder zurück zum eigentlichen Beweis. Durch Einsetzen von Gl. (9) in Gl. (10) erhält man

$$G(K) = \frac{1}{K} \sum_{i=1}^{N} \sum_{l=1}^{K} x_i^l \frac{1}{\prod_{j=0}^{K-l} \lambda(j)}. \tag{11}$$

Setzt man in Gl. (8) für den Zähler Gl. (9) und für den Nenner Gl. (11) ein, so bekommt man

$$\lambda(K) = \frac{1}{\prod_{j=0}^{K-1} \lambda(j)} \left[\frac{1}{K} \sum_{i=1}^{N} \sum_{l=1}^{K} x_i^l \frac{1}{\prod_{j=0}^{K-l} \lambda(j)} \right]^{-1}. \tag{12}$$

Das Produkt von $j = 0, \ldots, K-1$ über $\lambda(K)$ kann unter die Summenzeichen gezogen werden:

$$\lambda(K) = \frac{K}{\sum_{i=1}^{N} \sum_{l=1}^{K} x_i^l \left[\prod_{j=0}^{K-1} \lambda(j) / \prod_{j=0}^{K-l} \lambda(j) \right]} \tag{13}$$

Die ersten $(K-l)$ Faktoren der Produkte fallen durch Kürzungen weg:

$$\lambda(K) = \frac{K}{\sum_{i=1}^{N} \sum_{l=1}^{K} x_i^l \sum_{j=K-l+1}^{K-1} \lambda(j)} \tag{14}$$

Damit erhält man Gl. (3) für $m_i = 1$.

ii) Für $m_i \geq 1$:
Nach [13] gilt:

$$G(K) = \frac{1}{K} \sum_{i=1}^{N} \sum_{l=1}^{K} l\, p_i(l/K)$$

Durch Einsetzen der Gl. (14) in Gl. (9) und Umformung erhält man Gl. (3) für $m_i \geq 1$.

iii) Für $m_i = \infty$:
In diesem Fall gilt:

$$\bar{t}_i = \frac{1}{\mu_i},$$

d.h. mittlere Verweilzeit = mittlere Bedienzeit.
Nach Littles Gesetz gilt:

$$\bar{t}_i = \frac{e_i \lambda(K)}{\mu_i}$$

Durch Summierung beider Seiten über $i = 1, \ldots, N$ und Berücksichtigung, daß die Anzahl der Aufträge in einem geschlossenen Netz konstant ist, erhalten wir dann Gl. (3) für $m_i = \infty$. q.e.d.

Aus Gl. (3) für den Durchsatz können alle anderen Leistungsgrößen berechnet werden:

Die mittlere Anzahl von Aufträgen:

$$\bar{k}_i(K) = \sum_{l=0}^{K-1} \prod_{j=0}^{l} \lambda(K-j)\, x_i^{l+1} \tag{15}$$

Die Zustandswahrscheinlichkeiten:

$$p(k_1, \ldots, k_N) = \prod_{i=1}^{N} x_i^{k_i} \prod_{l=0}^{k_i} \lambda(k_i - l) \tag{16}$$

Dies folgt direkt aus der Berücksichtigung der Gl. (9).

4 Approximation von TUPAL

Eine wesentliche Rechenzeitverkürzung erreicht man, wenn man auf exakte Lösungen verzichtet und sie nur approximiert. Dabei nutzt man die Tatsache, daß der Durchsatz in Abhängigkeit von der Anzahl von Aufträgen im System gegen einen Maximalwert konvergiert (siehe Bild 2).

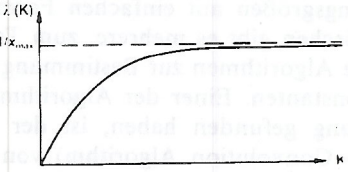

Bild 2.

Dieser Maximalwert $\lambda(K) = 1/x_{max}$ mit $x_{max} = \max(x_i)$, für $i = 1, \ldots, N$ wird im allgemeinen schon nach relativ wenigen Iterationen hinreichend genau erreicht, so daß die Berechnung dann abgebrochen werden kann. Man beginnt die Iteration mit $k = 1$ und bestimmt immer wieder den Durchsatz für die nächst höhere Anzahl von

Aufträgen im System. Nach jeder Iteration wird geprüft, ob

$$|\lambda(k)-\lambda(k-1)|<\varepsilon.$$

Sobald die Terminierungsbedingung erfüllt wird, bricht der Algorithmus ab. Die Durchsätze für die höhere Anzahl von Aufträgen werden dann dem letzten Durchsatzwert gleichgesetzt. Die Erfahrungen haben gezeigt, daß die dadurch erhaltenen Ergebnisse sehr gut sind. Sie unterscheiden sich von den exakten Werten meist erst in der dritten Stelle nach dem Komma.

5 Bewertung

Der Algorithmus TUPAL und seine Approximation wurden auf der Rechenanlage VAX 11/780 in der Programmiersprache PASCAL implementiert. Der Algorithmus wurde mit den bekannten herkömmlichen Methoden wie z. B. Mittelwertanalyse und LBANC-Methode verglichen.

Es wurden ca. 30 numerische Beispiele durchgeführt und die Rechenzeiten gemessen. Bei den Testläufen wurden die CPU-Zeiten mit Hilfe einer internen Funktion in der Programmiersprache C gemessen. Diese Funktion liefert als Wert die aktuelle Uhrzeit in Form einer real-Größe zurück. Mit Hilfe von 2 Variablen im Programm kann die Uhrzeit an 2 Stellen bestimmt und die CPU-Zeit als Differenz dieser beiden Variablen berechnet werden.

Für die von den Algorithmen verbrauchte CPU-Zeit ergaben sich folgende Werte in ms:

K /Alg	TUPAL	LBANC	MVA
10	330	270	130
25	410	700	210
50	550	870	220
75	660	990	240
100	710	1010	300
125	920	...	330

Wie aus der Tabelle ersichtlich, ist die Mittelwertanalyse der schnellste Algorithmus. Der Grund dafür, daß TUPAL langsamer ist als die MVA, liegt vor allem darin, daß in TUPAL die Potenz (x_i^l) vorkommt. Das Vorkommen der Potenz erhöht die Rechenzeit beträchtlich, wenn in Programmiersprachen gearbeitet wird, die sie nicht als Standardroutine bieten, sondern als Unterprogramm verlangen wie bei PASCAL. Im Vergleich zu LBANC ist TUPAL schneller. Die LBANC-Methode liefert den Wert für die Normalisierungskonstante ab K = 125 nicht mehr, weil der Wert bereits für K = 100 bei etwa 3.6*10**31 lag.

Als Grundlage für den Speicherplatzbedarf wird die Anzahl der Speicherplätze angenommen, die als Zwischenspeicher maximal reserviert werden müßten.

- TUPAL $O(K_R)$
- LBANC $O\left(N \cdot \prod_{r=1}^{R} (K_r+1)\right)$
- MVA $O\left(N \cdot \prod_{r=1}^{R} (K_r+1)\right)$

Hinsichtlich des Speicherplatzbedarfes ist der neue TUPAL-Algorithmus den herkömmlichen Methoden überlegen.

Bei einem Vergleich des Speicherplatzbedarfes muß allerdings immer berücksichtigt werden, daß durch die unterschiedlichen Möglichkeiten der Programmierung mehr oder weniger Speicherplatz eingespart werden kann.

Literatur

[1] *Jackson, J.:* Jobshop-like Queueing Systems. Management Science 10, 1, 1963, pp. 131–142.

[2] *Gordon, W.; Newell, G.:* Closed Queueing Systems with Exponential Servers. Operations Res. 15, 1967, pp. 254–265.

[3] *Baskett, F.; Chandy, K.; Muntz, R.; Palacios, G.:* Open, Closed and Mixed Network of Queues with Different Classes of Customers. JACM, Vol. 22, 2, Apr. 1975, pp. 248–260.

[4] *Muntz, R. R.:* Poisson Departure Process and Queueing Networks. 7th Ann. Princeton Conf., March 1973, pp. 435–440.

[5] *Chandy, K. M.:* The Analysis and Solutions for General Queueing Networks. 6th Ann. Princeton Conf. March 1972, pp. 224–228.

[6] *Chandy, K.; Howard, J. H.; Towsley, D.:* Product Form and Local Balance in Queueing Networks. North-Holland Publ. Co., Oct. 1976, pp. 89–103.

[7] *Chandy, K.; Martin, J.:* A Characterization of Product-Form Queueing Networks. JACM, Vol. 30, Nr. 2, April 1983, pp. 286–299.

[8] *Spirn, J. P.:* Queueing Networks with Random Selection for Service. IEEE, T-SE 5, 1979, pp. 287–289.

[9] *Noetzel, A. S.:* A Generalized Queueing Discipline for Product Form Network Solutions. JACM, Vol. 26, Nr. 4, Oct. 1979, pp. 779–793.

[10] *Buzen, J. P.:* Queueing Network Models of Multiprogramming. Ph. D. Thesis, Harvard Univ., Mass., Aug. 1971.

[11] *Chandy, K.; Herzog, U.; Woo, L.:* Parametric Analysis of Queueing Network Models. IBM Journal Res. Dev. 19, 1, Jan. 1975, pp. 36–42.

[12] *Wong, J. W.:* Queueing Network Models of Computer Systems. Ph. D. Thesis, UCLA-Eng-7579, June 1975.

[13] *Sauer, C.; Chandy, K.:* Computer Systems Performance Modelling. Prentice Hall, Englewood Cliffs, N.J., 1981.

[14] *Reiser, M.; Lavenberg, S. S.:* Mean Value Analysis of Closed Multichain Queueing Networks. JACM, Vol. 27, Nr. 2, Apr. 1980, pp. 313 to 322.

[15] *Bolch, G.; Akyildiz, I. F.:* Analyse von Rechensystemen. Teubner Verlag, 1982.

[16] *Akyildiz, I. F.:* Leistungsanalyse von Multiprozessorsystemen mit Prozeßkommunikation. Diss. am IMMD IV der Univ. Erlangen-Nürnberg, 1984.