<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Improving scheduling efficiency for high-speed routers with optical switch fabrics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Wu, B; Yeung, KL; Wang, X</td>
</tr>
<tr>
<td><strong>Citation</strong></td>
<td>Proceedings of the Global Telecommunications Conference, 2006 (GLOBECOM 2006), San Francisco, CA, USA, 27 November - 1 December 2006</td>
</tr>
<tr>
<td><strong>Issued Date</strong></td>
<td>2006</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10722/99262">http://hdl.handle.net/10722/99262</a></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>©2006 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.; This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.</td>
</tr>
</tbody>
</table>
Improving Scheduling Efficiency for High-Speed Routers with Optical Switch Fabrics

Bin Wu, Kwan L. Yeung and Xin Wang
Dept. of Electrical and Electronic Engineering
The University of Hong Kong, Pokfulam, Hong Kong
E-mail: {binwu, kyeung, xinwang}@eee.hku.hk

Abstract—Aiming at providing 100% throughput with bounded packet delay, we consider traffic scheduling in high-speed routers with optical switch fabrics. Because of the switch reconfiguration overhead, a speedup in the switch fabric is essential. For a given packet delay bound, our objective is to minimize the overall speedup $S=\delta_{\text{reconfig}}\times\delta_{\text{schedule}}$ so as to lower the implementation cost. Leveraging on the existing ADAPTIVE and DOUBLE algorithms, we show the speedup can be reduced by improving scheduling efficiency. Specifically, following the traffic matrix decomposition in ADAPTIVE and DOUBLE, we shift some packets from the residue matrix $R$ to the quotient matrix $Q$, while keeping the number of configurations required to cover each matrix the same. We reduce the number of time slots required to send the diminished residue matrix. In case of DOUBLE, this translates into a 12.5% cut in $S_{\text{schedule}}$ (from 2 to 1.75). We call the resulting algorithm Scheduling Residue First (SRF).

Keywords—Performance guaranteed switching; reconfiguration overhead; scheduling Residue First (SRF); speedup.

I. INTRODUCTION

Recent progress on optical switching technologies [1-2] has enabled the implementation of high-speed routers with optical switch fabrics as shown in Fig. 1. The optical switch fabric provides huge switching capacity as demanded by the backbone routers in the Internet. Since the input/output modules are connected with the central switch fabric by optical fibers, they can be distributed into several racks. As a result, power consumption can be reduced for each rack. This makes the resulting switch architecture more scalable.

On the other hand, optical switch fabric usually needs a non-negligible amount of reconfiguration overhead time to change its switch configuration from one to another, as well as to synchronize the optical signals arriving at the input ports [3]. During this period, no packet can be transmitted across the switch fabric. To achieve performance guaranteed switching (i.e. 100% throughput with bounded packet delay) [4-6], the switch fabric must run faster to compensate for both the reconfiguration overhead and the scheduling inefficiency. The required speedup $S$ is defined as the ratio of the internal packet transmission rate to the external line-rate ($S\geq 1$).

Assume time is slotted and each time slot can accommodate one fixed-size packet. Based on the $N\times N$ unicast switch in Fig. 1, several scheduling algorithms are proposed to achieve performance guaranteed switching [4-6]. They all adopt the same four-stage scheduling procedure as shown in Fig. 2. Stage 1 is for traffic accumulation. A traffic matrix $C(T)=\{c_{ij}\}$ is obtained at the input buffers every $T$ time slots. Each entry $c_{ij}$ denotes the number of packets arrived at input $i$ and destined to output $j$. As a common assumption in [4-6], the entries in each line (i.e. row or column) of $C(T)$ sum to at most $T$. In Stage 2, a scheduling algorithm generates a schedule consisting of (at most) $N_S$ switch configurations in $H$ time slots. Each configuration is denoted by a permutation matrix $P_n=[p^{(n)}_{ij}]$ ($N_S\geq H\geq 1$), where $p^{(n)}_{ij}=1$ means that input port $i$ is connected to output port $j$ (In this case, we also say that $P_n$ covers entry $(i, j)$). A weight $\phi_n$ is assigned to each $P_n$, indicating the number of slots that $P_n$ should be kept for packet transmission. The set of $N_S$ configurations generated must cover $C(T)$, i.e. $\sum_{n=1}^{N_S} \phi_n$, where $p^{(n)}_{ij}=\sum_{n=1}^{N_S} \phi_n$. Then $\sum_{n=1}^{N_S} \phi_n$ is the number of slots required to transmit all the packets in $C(T)$. Let each reconfiguration take an overhead of $\delta$ time slots. Accordingly, sending $C(T)$ requires $\delta N_S+\sum_{n=1}^{N_S} \phi_n$ time slots. This is generally larger than the traffic accumulation time $T$. Without speedup, 100% throughput is not possible. Stage 3 is for actual packet transmission, where the switch fabric is reconfigured according to the $N_S$ configurations. At a speedup of $S$, the slot size for a single packet transmission in Stage 3 is shortened by $S$ times. Then 100% throughput is ensured by having

$$\delta N_S+\frac{1}{S}\sum_{n=1}^{N_S} \phi_n = T.$$ (1)

The values of $N_S$ and $\sum_{n=1}^{N_S} \phi_n$ in (1) are determined by the scheduling algorithm. Note that the total reconfiguration
overhead time \( \delta N_S \) cannot be reduced by speedup and thus \( T > \delta N_S \). Finally, Stage 4 takes another \( T \) time slots to send the packets onto the output lines from the output buffers.

Rearranging (1), we have

\[
S = \frac{1}{T - \delta N_S} \sum_{n=0}^{N_S} \phi_n \times S_{\text{reconfigure}} \times S_{\text{schedule}},
\]

where \( S_{\text{reconfigure}} \) and \( S_{\text{schedule}} \) are defined as

\[
S_{\text{reconfigure}} = \frac{T}{T - \delta N_S} \tag{3}
\]

\[
S_{\text{schedule}} = \frac{1}{T} \sum_{n=0}^{N_S} \phi_n \tag{4}
\]

\( S_{\text{reconfigure}} \) is the speedup factor to compensate for the idle time caused by reconfigurations, whereas \( S_{\text{schedule}} \) is the speedup factor to compensate for the scheduling inefficiency.

In Fig. 2, packet delay is bounded by \( 2T+H \) slots and \( T > \delta N_S \). With a smaller \( N_S \), \( T \) and the packet delay bound can be reduced. But \( N_S \) must be no less than \( N \). Otherwise, the \( N_S \) configurations are not sufficient to cover every entry in \( C(T) \) [4, 6]. Besides, an algorithm with a smaller \( N_S \) generally has poorer scheduling efficiency, and requires a larger \( S_{\text{schedule}} \).

Among the existing scheduling algorithms, MIN [4] and QLEF [6] use the minimum number of \( N_S=2N \) configurations for scheduling, but the corresponding \( S_{\text{schedule}} \) is very large. On the other hand, EXACT [4] uses \( N_S=2N^2 \) configurations to get \( S_{\text{schedule}}=1 \), but the packet delay is very large due to the large value of \( N_S \). DOUBLE [4] makes an efficient tradeoff by using \( N_S=2N \) to achieve \( S_{\text{schedule}}=2 \). Note that those algorithms can only produce schedules with one of the three \( N_S \) values, \( N, N^2/2 \), or \( 2N \). Recently, ADAPTIVE [5] is proposed to allow any integer value of \( N_S \) in \( (N, N^2/2, 2N) \). It is also shown in [5] that DOUBLE is a special case of ADAPTIVE at \( N_S=2N \).

In this paper, we show that the schedules returned by ADAPTIVE can be further optimized. A new scheduling algorithm SRF (Scheduling Residue First) is proposed to improve the scheduling efficiency (i.e. to reduce \( S_{\text{schedule}} \)). In Section II, we review the two closely related algorithms DOUBLE [4] and ADAPTIVE [5], based on which our SRF is designed in Section III. Performance analysis and discussions are given in Section IV. We conclude the paper in Section V.

II. DOUBLE AND ADAPTIVE ALGORITHMS

We divide \( C(T) \) by \( T/(N_S-N) \) to get a quotient matrix \( Q=\{q_{ij}\} \) and a residue matrix \( R=\{r_{ij}\} \) as follows. If \( T/(N_S-N) \) is not an integer, use \( \lfloor T/(N_S-N) \rfloor \) instead [5].

\[
C(T) = \frac{T}{N_S - N} \times Q + R \tag{5}
\]

Since the entries in each line of \( C(T) \) sum to at most \( T \), the maximum line sum of \( Q \) is \( N_S-N \). Therefore, we can apply edge-coloring [7] to the bipartite multigraph of \( Q \) and get \( N_S-N \) colors/configurations to cover \( Q \) [5]. On the other hand, each entry in \( R \) is smaller than \( T/(N_S-N) \). So, \( R \) can be covered by any \( N \) non-overlapping configurations (i.e. any two of them do not cover the same entry), with each weighted by \( T/(N_S-N) \). As a result, \( C(T) \) can be covered by \( N_S-N+T/(N_S-N) \) configurations, each equally weighted by \( \phi_n = T/(N_S-N) \). From (4), we have

\[
S_{\text{schedule}} = \frac{1}{T} \sum_{n=0}^{N_S} \phi_n = \frac{T}{T - \delta N_S} = 1 + \frac{N}{N_S - N} \tag{6}
\]

Replacing \( N_S \) by \( 2N \) in (6), we get \( S_{\text{schedule}}=2 \) for DOUBLE [4]. Each of the matrices \( Q \) and \( R \) in DOUBLE is covered by \( N \) configurations, with an equal weight \( \phi_n = T/(N_S-N) \) for each configuration. Fig. 3 gives an example of DOUBLE execution.

Unlike DOUBLE, ADAPTIVE substitutes (6) into (2), and finds a proper \( N_S \) to minimize the overall speedup \( S \) by solving

\[
\frac{\partial S}{\partial N_S} = 0. \tag{7}
\]

III. SRF ALGORITHM

Since DOUBLE is a special case of ADAPTIVE at \( N_S=2N \), for simplicity, we first design SRF based on DOUBLE.

A. Observation and Motivation

From (5), we get \( C(T)=[T/(N_S-N)] \times Q + R \) with DOUBLE. For any \( r_{ij} \in R \), we have \( r_{ij} < T/N \). If \( r_{ij} \geq T/(2N) \), we call it an LER (large entry in \( R \)). Otherwise it is a SER (small entry in \( R \)). We have the following Lemma 1 (proved in the Appendix).

**Lemma 1:** In DOUBLE, if a line (row \( i \) or column \( j \)) in \( R \) contains \( k \) LERs, then in \( Q \) we have

\[
\sum_{i=1}^{N} q_{ij} \leq N - \left\lfloor \frac{k}{2} \right\rfloor \text{ for row } i, \text{ or } \sum_{j=1}^{N} q_{ij} \leq N - \left\lfloor \frac{k}{2} \right\rfloor \text{ for column } j.
\]

Based on Lemma 1, we can move some packets from \( R \) to \( Q \), while still keeping the maximum line sum of \( Q \) not more than \( N \). Note that all the weights in DOUBLE are equal and \( \phi_n = T/N \). So, if a line in \( R \) contains \( k \) LERs, we can move half (i.e. [\( k/2 \)]) of them to \( Q \) by setting them to 0 in \( R \), and at the same time increasing the corresponding entries in \( Q \) by one. Fig. 4 gives an example based on the \( Q \) and \( R \) in Fig. 3. We use \( Q' \) and \( R' \) to denote the updated \( Q \) and \( R \). Because the maximum
If the 4-tuple is a DD tuple, we set the two entries in a DD tuple always contains dominant (or equal) number of values as shown in Fig. 6. The tuples in Figs. 6a~6l are said to be diagonal dominant (DD), and the two circled entries are defined as dominant entries. Each of the two dominant entries in a DD tuple always contains dominant (or equal) number of 1s than both of its line images. On the contrary, the non-DD tuples in Figs. 6m~6p do not have such a property. Each non-DD tuple in Figs. 6m~6n is called a column isomorphic tuple because the two columns are exactly the same. Similarly, the non-DD tuples in Figs. 6o~6p are row isomorphic tuples.

To cover the residue matrix in N configurations $P_1\sim P_N$, we first initialize them to all-0s. Based on reference matrix $F$, each PPC is sequentially checked to construct two configurations $P_p$ and $P_{p+N/2}$. Particularly, for each $f_j$ covered by PPC$\phi$, we find the 4-tuple $\{f_p,f_{(N+1)},f_{(N+2)},f_{(N+3)}\}$ may have 16 possible values as shown in Fig. 7. The tuples in Figs. 7a~7d are said to be diagonal dominant (DD), and the two circled entries are defined as dominant entries. Each of the two dominant entries in a DD tuple always contains dominant (or equal) number of 1s than both of its line images. On the contrary, the non-DD tuples in Figs. 7m~7p do not have such a property. Each non-DD tuple in Figs. 7m~7n is called a column isomorphic tuple because the two columns are exactly the same. Similarly, the non-DD tuples in Figs. 7o~7p are row isomorphic tuples.

To cover the residue matrix in N configurations $P_1\sim P_N$, we first initialize them to all-0s. Based on reference matrix $F$, each PPC is sequentially checked to construct two configurations $P_p$ and $P_{p+N/2}$. Particularly, for each $f_j$ covered by PPC$\phi$, we find the 4-tuple $\{f_p,f_{(N+1)},f_{(N+2)},f_{(N+3)}\}$ may have 16 possible values as shown in Fig. 7. The tuples in Figs. 7a~7d are said to be diagonal dominant (DD), and the two circled entries are defined as dominant entries. Each of the two dominant entries in a DD tuple always contains dominant (or equal) number of 1s than both of its line images. On the contrary, the non-DD tuples in Figs. 7m~7p do not have such a property. Each non-DD tuple in Figs. 7m~7n is called a column isomorphic tuple because the two columns are exactly the same. Similarly, the non-DD tuples in Figs. 7o~7p are row isomorphic tuples.

To cover the residue matrix in N configurations $P_1\sim P_N$, we first initialize them to all-0s. Based on reference matrix $F$, each PPC is sequentially checked to construct two configurations $P_p$ and $P_{p+N/2}$. Particularly, for each $f_j$ covered by PPC$\phi$, we find the 4-tuple $\{f_p,f_{(N+1)},f_{(N+2)},f_{(N+3)}\}$ may have 16 possible values as shown in Fig. 7. The tuples in Figs. 7a~7d are said to be diagonal dominant (DD), and the two circled entries are defined as dominant entries. Each of the two dominant entries in a DD tuple always contains dominant (or equal) number of 1s than both of its line images. On the contrary, the non-DD tuples in Figs. 7m~7p do not have such a property. Each non-DD tuple in Figs. 7m~7n is called a column isomorphic tuple because the two columns are exactly the same. Similarly, the non-DD tuples in Figs. 7o~7p are row isomorphic tuples.
\[ \begin{pmatrix} f_{ij} & f_{ij+N} & f_{ij+(N+1)} \end{pmatrix} \]

\[ f_{ij} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \]

Fig. 6. Sixteen possible combinations of \( \{ f_{ij}, f_{ij+N}, f_{ij+(N+1)} \} \).

\[ F = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \]

Fig. 7. Residue matrix scheduling based on \( F \).

\begin{align*}
\text{SRF Algorithm} \\
\text{Input:} & \quad \delta \text{ and an } N \times N \text{ matrix } C(T) \text{ with maximum line sum not more than } T. \\
\text{Output:} & \quad N \text{ configurations } P_1, \ldots, P_N \text{ and weights } \phi_1, \ldots, \phi_N. \\
\text{Step 1: Divide } C(T): \\
\text{Calculate } N_0 \text{ by (11). Use } \left[ T[N_0 \times N_0] \right] \text{ to divide } C(T) \text{ and separate it into } Q = [q_{ij}] \text{ and } R = [r_{ij}] \text{ as in (5).} \\
\text{Step 2: Schedule the residue matrix:} \\
a) & \quad \text{Define a reference matrix } F = \{ f_{ij} \} \text{ based on } R, \text{ where } f_{ij} = 1 \text{ if } r_{ij} \text{ is an \( \text{LER} \) or } j \text{ otherwise. Define } N_2 \text{ PPCs based on (9). Initialize } P_1 \sim P_N \text{ to all-0 matrices. Set } 1 \rightarrow p. \\
b) & \quad \text{Pick up an entry } f_{ij} \text{ covered by } PPC_0 \text{, find its images to form a 4-tuple } \{ f_{ij}, f_{ij+N}, f_{ij+(N+1)}, f_{ij+N+(N+1)} \}. \text{ If it is a DD tuple, set the two dominant entries to } 1 \text{ in } P_R \text{ and the other two non-dominant entries to } 1 \text{ in } P_{NR}. \text{ If it is an isomorphic tuple, check whether another isomorphic tuple occurred in examining PPC and PPC}_2 \text{. Then invoke the butterfly mechanism to set the entries in } P_R \text{ such that the number of } 1s \text{ in this line pair of } F \text{ can be covered by } P_R \text{ and } P_P \text{ alternatively. If a pair of diagonal entries are set to } 1 \text{ in } P_R, \text{ then set the other pair of diagonal entries to } 1 \text{ in } P_{NR}. \text{ Repeat step } 2b) \text{ for each } f_{ij} \text{ covered by } PPC_0, \text{ until the two configurations } P_1 \text{ and } P_{NR} \text{ are obtained.} \\
c) & \quad \text{Set } p \rightarrow p+1. \text{ Repeat step } 2b) \text{ until } p = N_2 + 1. \\
d) & \quad \text{Set } \phi_1 \sim \phi_{N_2} \text{ to } [T[N \times N_0] \text{ and } \phi_{N_2 + 1} \rightarrow \phi_1 \text{ to } [T[N \times N_0]]. \text{ If some } 1s \text{ in } F \text{ are covered by } P_{NR} \sim P_N, \text{ increase the corresponding entries in } Q \text{ by one. Denote the updated } Q \text{ by } Q'. \\
\text{Step 3: Schedule the quotient matrix:} \\
a) & \quad \text{Construct a bipartite multigraph } G \text{ from } Q'. \text{ Find a minimal edge-coloring of } G \text{ to get at most } N_2 \text{ colors. Set } N_2+1 = n. \\
b) & \quad \text{Use the edge-coloring of } G \text{ to construct a configuration } P_0 \text{ from the edges assigned to that color. Set } \theta_0 = [T[N \times N_0]] \text{ and } n \rightarrow n+1. \text{ Repeat step } 3b) \text{ for each of the colors in } G. \\
\end{align*}

In SRF (see Fig. 8), we replace \( S_{\text{schedule}} \) in (2) by (10), and solve (7) to get the proper \( N_2 \) value, as given in (11).

\[ N_{S}^{\text{real}} \begin{cases} \frac{1}{4} \left( N + \sqrt{\frac{12NT}{\delta} - 3N^2} \right) & \text{if } N^2 - 2N + 2 > N_{S}^{\text{real}} \geq N + 1 \\ N = N + 1 & \text{if } N + 1 > N_{S}^{\text{real}} \end{cases} \]

Compared to ADAPTIVE [5] which has a time complexity of \( O((N^4 \log N)) \), SRF needs an extra \( O(N^3) \) computation for residue matrix scheduling. So the time complexity of SRF is \( O(N^{1.5} \log N + N^3) \), or \( O(N^3) \).
S\text{\_schedule} in (6) for ADAPTIVE and in (10) for SRF are compared in Fig. 9 for N=64. We did not directly compare the overall speedup S because it involves other parameters such as T and \(\delta\). From Fig. 9, again we can see that DOUBLE [4] is a special case of ADAPTIVE [5]. DOUBLE generates \(S_{\text{\_schedule}}=2\) at \(N_s=128\), but we can get \(S_{\text{\_schedule}}=1.75\) with SRF. This gives a cut of 12.5\% on \(S_{\text{\_schedule}}\). On the other hand, with \(S_{\text{\_schedule}}=2\), SRF only requires \(N_s=112\) configurations to schedule \(C(T)\).

A simple example is given in Fig. 10 to compare the three algorithms. Although DOUBLE produces the smallest \(S_{\text{\_schedule}}\) (but with the largest \(N_s\)), it has the largest overall speedup of \(S=14\), whereas ADAPTIVE has \(S=8.4\) and SRF has \(S=7\).

Note that SRF does not allow \(N_s=N\). But this case can be efficiently handled by the minimum delay scheduling algorithm QLEF [6]. Also note that predetermining PPCs in a cyclic manner using (9) is not necessary. We did so only to facilitate our presentation. In fact, any set of \(N/2\) non-overlapping permutation sub-matrices in \(Z_1\) can be used as PPCs.

V. CONCLUSION

We proposed a new algorithm SRF (Scheduling Residue First) to improve the scheduling efficiency for high-speed routers with optical switch fabrics. SRF significantly reduces the speedup factor \(S_{\text{\_schedule}}\). Compared with DOUBLE, a 12.5\% cut on \(S_{\text{\_schedule}}\) is achieved. Consequently, SRF outperforms the existing DOUBLE and ADAPTIVE algorithms by requiring a lower overall speedup for performance guaranteed switching.

APPENDIX

Lemma 1: In DOUBLE, if a line (row i or column j) in \(R\) contains \(k\) LERs, then in \(Q\) we have

\[
\sum_{j=1}^{N} q_j \leq N - \left\lfloor \frac{k}{2} \right\rfloor
\]

for row i, or

\[
\sum_{i=1}^{N} q_i \leq N - \left\lfloor \frac{k}{2} \right\rfloor
\]

for column j.

Proof: After \(C(T)\) is divided by \(T/N\), we have

\[
C(T) = \frac{T}{N} Q + R \quad \text{or} \quad c_y = \frac{T}{N} q_y + r_y.
\]

Assume row i of \(R\) contains \(k\) LERs. Because

\[
\sum_{j=1}^{N} c_y = \frac{T}{N} \sum_{j=1}^{N} q_j + \sum_{j=1}^{N} r_y \leq T,
\]

we have

\[
\sum_{j=1}^{N} q_j \leq N - \left\lfloor \frac{k}{2} \right\rfloor
\]

for row i, or

\[
\sum_{i=1}^{N} q_i \leq N - \left\lfloor \frac{k}{2} \right\rfloor
\]

for column j.

REFERENCES


Fig. 10. An example of DOUBLE, ADAPTIVE and SRF.