Collaboration and Multitasking in Networks: Aligning Task Priorities and Collaboration Levels

Itai Gurvich and Jan A. Van Mieghem
Kellogg School of Management, Northwestern University

March 30, 2015; Revised Sep. 19, 2015

Motivated by the trend towards more collaboration in team work, we study networks where some tasks require the simultaneous processing by multiple types of multitasking human or indivisible resources. The capacity of such networks is generally smaller than the bottleneck capacity. In Gurvich and Van Mieghem (2015) we proved that both capacities are equal in networks with a hierarchical collaboration architecture, which define a *collaboration level* for each task depending on how many types of resources it requires relative to other network tasks. This paper studies how task prioritization impacts the capacity of such hierarchical networks using a conceptual queuing framework that formalizes coordination and switching idleness.

To maximize the capacity of a team, highest priority should be given to the tasks that require the most collaboration. Otherwise, a mismatch between priority levels and collaboration levels inevitably inflicts a capacity loss. We demonstrate this essential trade-off between task prioritization and capacity in a basic collaborative network and in parallel networks. To manage this trade-off, we present a hierarchical threshold priority policy that balances switching and coordination idleness.

Key words: collaboration, architectures, resource sharing, multitasking, priority, teams, stability, control.

1. Introduction and Summary of Results

Motivated by the prevalence of collaborative processing in services, we study how simultaneous collaboration and multitasking impact system capacity and responsiveness. Simultaneous collaboration means that some tasks require the synchronous processing by multiple types of resources. Discharging a patient, for example, may require the presence of both a doctor and a nurse and chemical distilling requires two vessels. Multitasking means that a resource performs multiple activities. Multitasking is equivalent to resource sharing and means that multiple activities require the same resource. A doctor, for example, may be required for both patient diagnosis and for patient discharge. Simultaneous collaboration imposes constraints on the capacity of the process because *multitasking* resources have to be *simultaneously* at the right place. Human operators magnify the effect of these synchronization requirements because they are indivisible and cannot be "split." An emergency physician may split her time among multiple activities—spending $x\%$ of her time in one activity and the remaining $(100-x)\%$ in another—and she may switch between activities frequently, yet she cannot process two activities at the same time (which may, in this example, require her physical presence in two distinct locations).

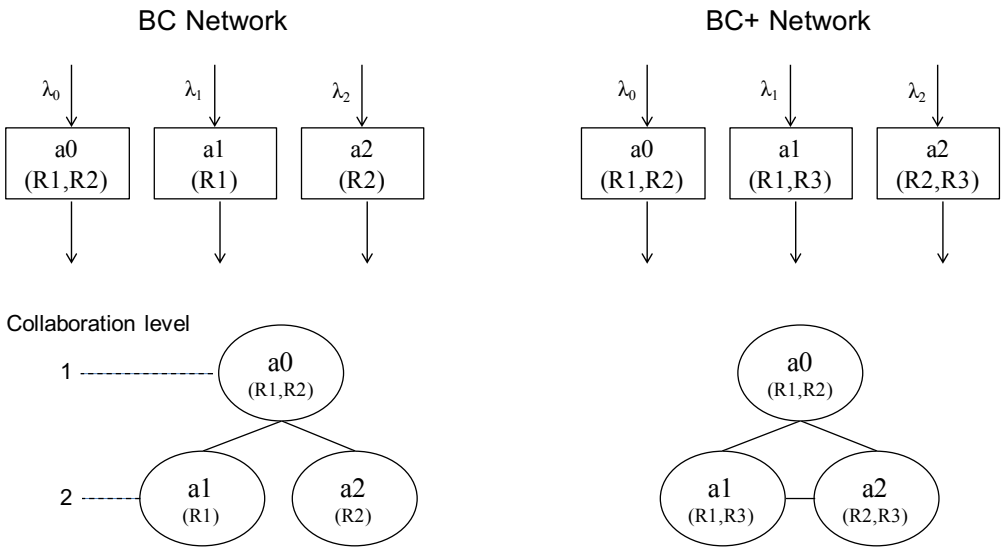

Figure 1 (LEFT) The Basic Collaboration (BC) network with a hierarchical collaboration architecture. (RIGHT) Adding a third resource yields the BC+ network: the prototypical non-hierarchical network.

Collaboration of indivisible multitasking resources poses significant operational and interesting intellectual challenges. In general, the capacity of such networks is smaller than the bottleneck capacity. Consider, for example, the basic collaboration (BC) network on the left of Figure 1 with three activities ($a0$, $a1$ and $a2$) and two multitasking resources (R1 and R2). Only resource $j \in \{1, 2\}$ is required for its *individual* activity ai while both resources are needed for the *collaborative* task $a0$. Given mean arrival rate λ_i and mean service time m_i for activity $i \in \{0, 1, 2\}$, the load on resource j is

$$\rho_j = \lambda_0 m_0 + \lambda_j m_j.$$

Resource j is a bottleneck if $j \in \arg\max\{\rho_1, \rho_2\}$ and the bottleneck load is

$$\rho^{\mathrm{BN}} = \max\{\rho_1, \rho_2\}.$$

Processing networks can typically 'handle any arrival rates' (while keeping the system stable) as long as $\rho^{\mathrm{BN}} < 1$, but that need not be true when indivisible multitasking resources collaborate. To see what can go wrong, consider the BC+ network in the right panel of Figure 1 where we added resource 3. With this resource we also added a collaboration dependency. The network must, at any given time, use one of only three *feasible configurations vectors* that specify which activities are "active." As no two activities can be performed in parallel, only the unit vectors are feasible. For an arrival rate vector to be sustainable there must exist time allocations π_1, π_2, π_3 such that

$$\pi_0 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \pi_2 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \pi_3 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} m_1\lambda_1 \\ m_2\lambda_2 \\ m_3\lambda_3 \end{pmatrix}$$

for some $\pi \geq 0$ with $\sum_i \pi_i \leq 1$. The *network* utilization $\rho^{\text{net}}(\lambda) := \sum_i \pi_i$ can be interpreted as the average time it takes the network to process an input λ. This will in general exceed the time, ρ^{BN}, it takes the bottlenecks, working in isolation, to process that input. If $m_j \equiv 1$ and $1/3 < \lambda_1 = \lambda_2 = \lambda_3 < 1/2$, then $\rho^{\text{BN}} < 1$ yet $\rho^{\text{net}}(\lambda) > 1$ and the queues at each task would explode. Thus, collaboration and multitasking can inflict a capacity loss when resources are indivisible.[1]

The BC and BC+ networks fundamentally differ in their underlying *collaboration architecture*. A network's architecture specifies how resources are assigned to activities and can be represented by a collaboration graph that has a node for each activity and an edge between two nodes if they require overlapping resource sets. Hierarchical collaboration architectures can be represented by a graph where nodes are arranged in "collaboration levels" and resources at each node are a subset of those used by the node above it, as Figure 1 shows for the BC network. In contrast, the BC+ collaboration architecture is not hierarchical.

In Gurvich and Van Mieghem (2015) (further abbreviated to GVM) we prove that networks with hierarchical architectures do not feature *Unavoidable Bottleneck Idleness*: $\text{UBI}(\lambda) = \rho^{\text{net}}(\lambda) - \rho^{\text{BN}}(\lambda) = 0$ for each $\lambda \geq 0$. UBI is caused by the indivisibility of resources that prevent fractional capacity allocations. In hierarchical networks, only integer allocations are optimal and hence indivisibility is costless (UBI = 0).

In this paper, we study how the theoretical network capacity is achieved through, and affected by, task prioritization policies in the BC network and in a larger class of parallel networks with hierarchical collaboration architecture. To conceptualize the challenges and tradeoffs we introduce two types of *collaboration* idleness that impact capacity more subtly than UBI.

Task prioritization requires synchronized (joint) movements of resources between tasks. It is useful to think of the control rule setting an *alarm* that signals to resources when to switch. In the BC network, such alarm signals when resources should move from the individual tasks to the collaborative task, and back. The network incurs *coordination idleness* when a resource is idly waiting for the alarm to move to a collaborative task that has work—it is waiting for a coordinated move. In the BC network, resource 1 might not have work in activity $a1$ but, depending on the priority rule, might have to wait until the alarm sounds before moving to $a0$ where there could be work. The network incurs *switching idleness* when a resource is idly waiting after the alarm has sounded but before moving to the collaborative task. This happens only under non-preemptive controls when a resource has finished its service but is waiting for the remaining required resources to finish theirs. (These two types of idleness are formalized in §6.) In contrast to UBI,

[1] If resources were divisible, allocating a fraction of each resource to each of its activities would split the network into three independent queues and there would be no capacity loss in the BC+ network.

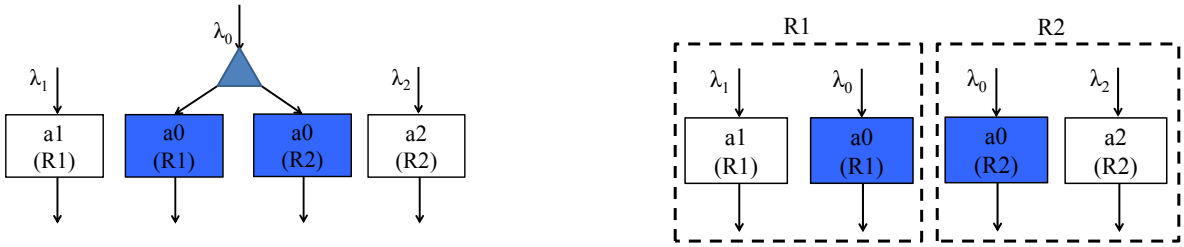

Figure 2 The BC network of Fig. 1 can be relaxed by duplicating the arrivals to the collaborative task (LEFT) to yield the benchmark (Two-class $M/M/1)^2$ network (RIGHT).

coordination and switching idleness depend not only on the architecture but also on the control policy. Essentially, the idleness that a control imposes on the network is:

$$\text{Collaboration idleness} = \text{Coordination idleness} + \text{Switching idleness}.$$

It is instructive to introduce the benchmark system obtained from the BC network by duplicating the arrival stream to the collaborative task (left panel in Fig. 2). This breaks the collaboration constraint and allows the resources to work on the collaborative jobs separately. Effectively, each resource sees a two-class $M/M/1$ queue (right panel). The performance gap between the benchmark and the BC network represents the "cost" of collaboration. The benchmark system exhibits four fundamental phenomena ("congestion laws") in well-operated traditional queuing networks: First, there exists a control policy that stabilizes the system for any arrival rate for which $\rho^{\rm BN} < 1$. Second, the total workload scales like $1/(1-\rho^{\rm BN})$ as utilization $\rho^{\rm BN} \uparrow 1$. Indeed, under any work conserving policy, the steady-state total number $Q_+(\infty)$ in an M/M/1 system is of order $\mathcal{O}\left(1/(1-\rho)\right)$ because

$$\mathbb{E}Q_+(\infty) = \frac{\rho}{1-\rho}.$$

Third, this scaling rate can be maintained while prioritizing certain queues over others. Indeed, whether one prioritizes queue 1 or 2 in a two-class M/M/1, the total workload follows the optimal scaling $\mathcal{O}\left(1/(1-\rho)\right)$ as long as the policy is work conserving. We call this *controllability*: the freedom to prioritize certain queues over others without significantly compromising the performance of the total queue count or workload. And fourth, non-preemptive (NP) policies carry no significant cost over preemptive (P) policies. Indeed, the impact of preemptive versus non-preemptive priority service on queue lengths in a two-class M/M/1 system is small, as shown in Figure 3. In particular, it is negligible relative to $1/(1-\rho)$: As $\rho \uparrow 1$, the high priority queues remain finite under either discipline—and hence, when scaled by $(1-\rho)$ both approach 0—while the low priority queues hold all the work and are $\mathcal{O}\left(1/(1-\rho)\right)$. We will show that controllability and the near-equivalence between preemptive and non-preemptive priority can break down in collaborative networks.

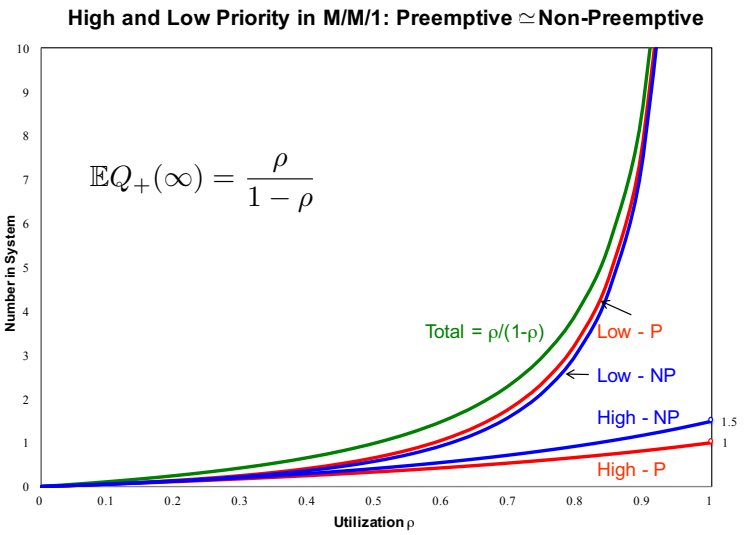

Figure 3 **The two-class $M/M/1$ queue illustrates the four "congestion laws." (For $\lambda_1 m_1 = \lambda_2 m_2 = \rho/2$)**

The following summarizes our findings in the BC network and in parallel networks with collaborating, multitasking, indivisible resources:

1. Misaligned task priorities and collaboration levels or non-preemptive priority service result in a capacity loss. Significant switching or coordination idleness destabilizes the network for values of $\rho^{\rm BN}$ strictly below 1. Typically there is a strong trade-off between priority control (waiting) and capacity. In the extreme that one does not prioritize at all, then we show that a polling policy where servers move only after they exhaust their own queue introduces significant coordination idleness that does not destabilize the system but leads to large queues of at least order $(1-\rho^{\rm BN})^{-3/2}$. We introduce a family of *hierarchical threshold priority policies* that can choose threshold levels to achieve any desired balance between priority control (waiting) and capacity. We explain this in terms of how thresholds impact switching and coordination idleness. When thresholds are large (i.e., of order $1/(1-\rho)$), full capacity is regained.

2. Aligning task priorities and collaboration levels is required to maximize capacity. This implies that highest priority should be given to the tasks that require the most collaboration. We present a family of *hierarchical priority policies* that stabilize, and achieve optimal queue scaling in, the BC network and parallel networks with indivisible resources up to full bottleneck capacity $\rho^{\rm BN} < 1$ if the collaboration architecture is hierarchical and preemption is allowed.

Our findings show that in these collaborating networks, one loses controllability (one cannot simply choose any subset of the queues as priority queues without losing significant throughput) and the difference between preemptive and non-preemptive service is fundamental and persists in heavy-traffic (which is new to the best of our knowledge).

The impact of (mis)matching task priorities and collaboration levels in the BC network is captured by the 2×2 matrix in Figure 4. In the BC network, the collaborative activity has highest

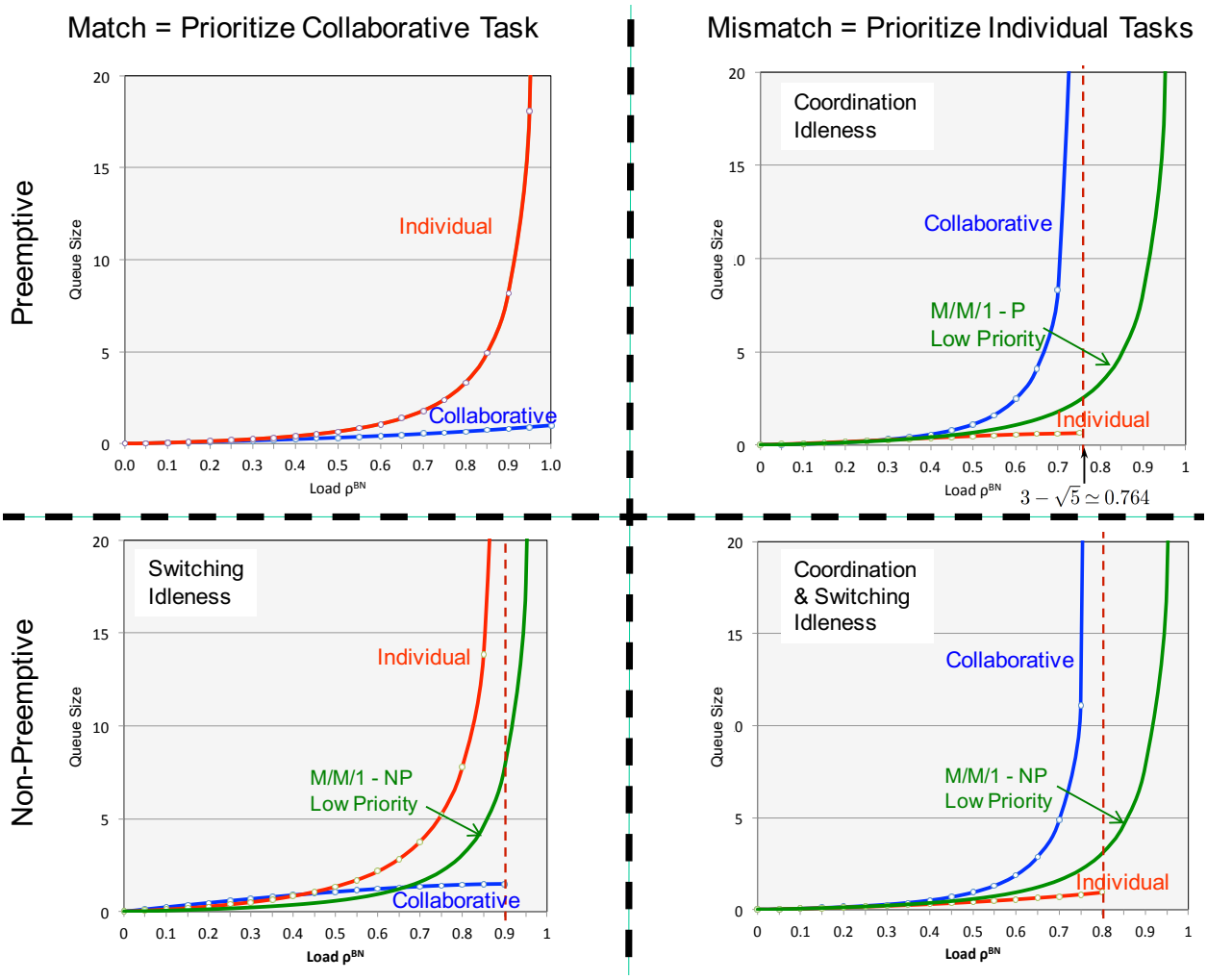

Figure 4 **How (mis)matched task prioritization affects capacity for preemptive and non-preemptive service.**

collaboration level. We contrast matched priority to the collaborative task with mismatched priority to both individual tasks and also compare to the two-class $M/M/1$ benchmark. While the plots use specific priority policies and parameters[2], we prove that these are universal properties (i.e., there is no control policy that can avoid these properties):

1. Matched preemptive priority to the collaborative task achieves maximal capacity and optimal scaling. This is the only quadrant in Figure 4 where collaboration comes at no cost. It also shows that the two-class $M/M/1$ benchmark is tight: it is achievable.

2. Matched non-preemptive priority to the collaborative task inflicts switching idleness and results in a significant capacity loss. In the simulation in Figure 4, throughput cannot exceed a bottleneck utilization of roughly 0.9. To generalize this observation we prove a tradeoff result: there exists **no** control policy that makes the collaborative queue short and keeps the total expected queue count finite as ρ^{BN} approaches 1.

[2] The plots show the average queue lengths during a simulation of 10 million time units where $m_i = 1/2$ and $\lambda_i = \rho$.

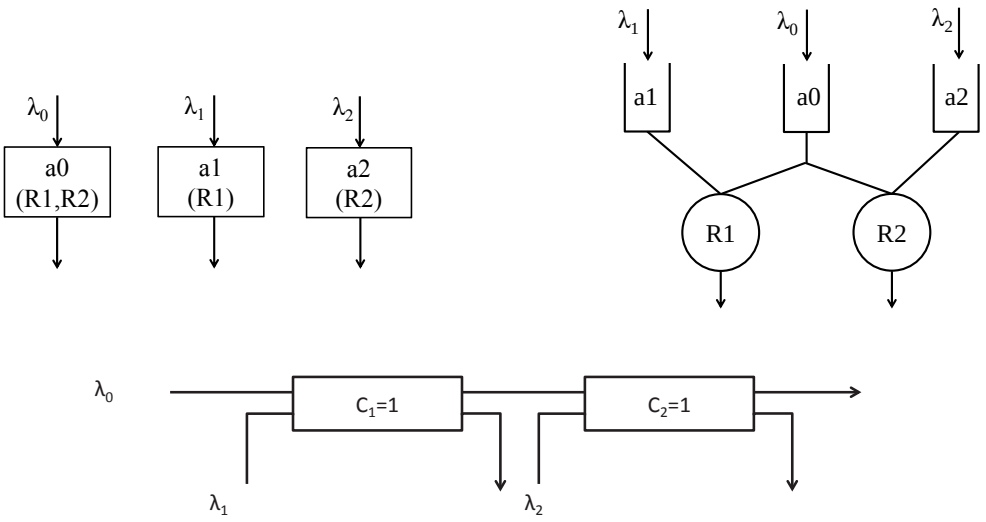

Figure 5 Different, but equivalent, views of the BC network: (TOP LEFT) Activity view; (TOP RIGHT) Resource view; (BOTTOM) Bandwidth Sharing view.

3. Mismatched preemptive priority to both individual tasks inflicts coordination idleness and results in capacity loss. We prove that the capacity in this example is only $3-\sqrt{5}=0.764$.

4. Mismatched non-preemptive priority to both individual tasks results in significant capacity loss. While the system now incurs both coordination *and* switching idleness, it performs slightly better than the preemptive case because the switching frequency is reduced.

In summary: Collaboration introduces a stark trade-off between task prioritization (service considerations) and capacity. Misaligned priority levels and collaboration levels inflict a significant capacity loss due to excessive switching or collaboration idleness. Matching priority and collaboration levels mitigates the trade-off and even eliminates it with preemptive service (top-left quadrant). The hierarchical priority threshold policy that we introduce balances coordination and switching idleness and traces the queue control vs capacity trade-off curve that we will show as our final figure when we conclude in section 8.

2. Literature review

A vast literature studies performance analysis and optimization of resource sharing in networks, often inspired by communication networks. Two strands are particularly relevant: the literature on switch networks and bandwidth sharing networks. To emphasize the connections, Figure 5 juxtaposes the different, but equivalent, pictorial network representations used in this paper versus typical papers on queuing networks and communication networks.

Switched networks. A network switch is a computer networking device that connects devices together on a computer network, by using packet switching to receive, process and forward data to

the destination device (Wikipedia 2015). An extensive literature studies switched networks from a queuing theoretic perspective. One can distinguish a standard input-queued switch (for a recent study, see e.g. Kang et al. (2012)) from more general switches; and single-hop switches in which packets depart the network after service at a queue from multihop networks, where packets depart after receiving service at one or more queues; see Shah and Wischik (2012). In a general single-hop switch, each of the J queues has its dedicated arrival stream. Arrivals occur at beginning of each time period with mean of λ_i for queue i. In the beginning of each period jobs are depleted from the queues. Processing takes one time unit exactly and the depletion of jobs is constrained by the so-called "schedules" which, as our configuration vectors, are binary vectors. That is,

$$Q(\tau+1) = [Q(\tau) - dB(\tau)]^+ + dA(\tau),$$

where $Q(\tau+1)$ is the queue (vector) at the beginning of period $\tau+1$, $dA(\tau)$ is the number of arrivals at the end of period τ and $dB(\tau) \in \mathcal{S}$ where $\mathcal{S}$ is the family of allowed schedules.

Within this general framework, a discrete-time version of the BC network is a switch with schedule set $\mathcal{S} = \{(1,0,0)', (0,1,0)', (0,0,1)', (0,1,1)'\}$. In fact, the discrete-time version of any collaborative network can be mapped to a switch and the set of schedules in the corresponding switch is identical to the family of configuration vectors. Thus, switches and (human) collaborative networks have in common the restriction to integral allocations (in contrast to bandwidth sharing networks–see more below).

Our collaborative networks differ from switched networks along two dimensions: First, our study requires continuous time for the distinction between preemptive policies and non-preemptive policies to be meaningful. Second, and perhaps more fundamental, switches are defined by their set of schedules and have no explicit associated set of resources. A frequently studied objective of switch control is throughput maximization and optimal growth of the **total** queue as both a function of ρ^{net} and the number of queues; see the recent Walton (2015) and Shah et al. (2014). In contrast, the model primitive in our study is *how* the various types of resources collaborate and the notion of their bottleneck capacity. The collaboration architecture results in the set of feasible configuration vectors (schedules) that define the network capacity. This allows us to study how aligning priorities with collaboration levels impacts the network capacity relative to the bottleneck capacity, questions that have no meaning in the general model of a switch.

A last important remark concerning switches: The widely cited and fundamental result of Stolyar (2004) shows that under a complete resource pooling assumption on the family of schedules, questions of switch controllability are in a strong sense resolved: one can minimize any strictly

convex holding cost without any compromise to throughput.[3] Importantly, however, the "switch-model" of the BC network (the simplest of collaborative networks) does not satisfy the complete resource pooling assumptions. This, again, illustrates the value of studying collaboration directly rather than as a special case of switches.

Bandwidth sharing networks transmit jobs ("traffic") over a set of capacitated, divisible resources ("links"). Multiple types of traffic can share a link and some types of traffic may simultaneously require multiple links in its transmission, which coincides with our notion of collaboration.

The critical difference between our study and bandwidth sharing networks is the indivisbility of our resources. Indeed, our first paper Gurvich and Van Mieghem (2015) starts from the simple observation that indivisibility leads inevitably to capacity loss relative to the capacity of a divisible resource sharing model (which equals the bottleneck capacity). Therefore, while the network representations in Fig. 5 are equivalent[4], the model assumptions are critically different and the research focus is different.

Queues with collaboration also turn out to be closely related to queues with switchover times and queues with interruptions. Borst (1996) covers performance analysis for a variety of polling systems with switchover times; for optimization and control see, e.g., Reiman and Wein (1998); Lan and Olsen (2006) and the references therein. While switchover times are typically exogenous parameters in the polling literature, they arise endogenously in collaborative networks. Some of our collaborative priority policies represent an "interruption" for the individual queue; see the survey by Krishnamoorthy et al. (2014).

Finally, our work also connects to the emerging operations literature on managing white collar work which involves indivisible (human) resources that often collaborate and multitask. In their survey, Hopp et al. (2009) observe that "white-collar work is much less understood in an operations sense than is blue-collar work. ... We do not yet have principles for guiding operations decisions. Fundamental questions remain unanswered. For example: What is the bottleneck of a white-collar work system?" Our paper investigates the subtlety of capacity of such systems.

3. Task Prioritization and Capacity Loss in the BC Network

In this section, we first show how a mismatch between task priority levels and collaboration levels inevitably leads to a capacity loss in the BC network. Recall that the collaborative activity has the highest collaboration level so a mismatch means prioritizing the individual activities. Then we show that matching priority and collaboration levels recovers theoretical capacity but only when service is preemptive.

[3] While Stolyar's result focuses on finite horizon optimality it is likely that his result extends to a stationary setting.

[4] The two-link network in Harrison et al. (2014) is the resource-splitting version of our BC network.

3.1. Mismatch: Prioritizing the Individual Tasks

First we consider giving static priority to *both* individual tasks, which means that both servers move to their individual task as soon as queue Q_1 or Q_2 is positive. We shall call this I^2-priority to contrast with I^1-priority which gives static priority to *one* specific individual task.

Proposition 1 (I^2 Priority Capacity Loss) *Static priority to both individual tasks reduces capacity. With preemptive I^2 priority, the BC network is unstable (not positive recurrent) if and only if*

$$\frac{\lambda_0}{\mu_0} \geq \left(1-\frac{\lambda_1}{\mu_1}\right)\left(1-\frac{\lambda_2}{\mu_2}\right). \tag{1}$$

With nonpreemptive I^2 priority, the BC network is unstable if

$$\frac{\lambda_0}{\mu_0} > \left(1-\frac{\lambda_1}{\mu_1}\right)\left(1-\frac{\lambda_2}{\mu_2}\right)\left[\frac{1+m_0(\lambda_1+\lambda_2)}{1+m_0(\lambda_1+\lambda_2)\left(1-\frac{\lambda_1}{\mu_1}\right)\left(1-\frac{\lambda_2}{\mu_2}\right)}\right]. \tag{2}$$

Both conditions imply a capacity loss: we cannot take ρ^{BN} close to 1 without making the system explode. Recall that the two-class M/M/1 system is unstable iff $\frac{\lambda_0}{\mu_0} \geq 1-\frac{\lambda_i}{\mu_i}$, which corresponds to a strictly larger capacity region than in the BC network with I^2 priority when λ_1 and λ_2 are both positive (excluding the degenerate case where one of them vanishes). Consider the numerical example shown in Fig. 4 for the symmetric BC network with $m_i = 0.5$: the maximal throughput $\lambda_i = \bar{\lambda}$ under preemptive I^2 priority solves $\bar{\lambda}/2 = (1-\bar{\lambda}/2)^2$ with solution $\bar{\lambda} = 3-\sqrt{5} \simeq 0.764$. This maximal $\rho^{\mathrm{BN}} = (m_0+m_1)\bar{\lambda} = 0.764$ represents a capacity loss of 23.6%. [5]

Non-preemptive I^2 priority regains some capacity but not all: Notice that the right hand side of the sufficient[6] condition (2) is strictly less than 1: since $\left(1-\frac{\lambda_1}{\mu_1}\right)\left(1-\frac{\lambda_2}{\mu_2}\right) < 1$, we have

$$1 < \left[\frac{1+m_0(\lambda_1+\lambda_2)}{1+m_0(\lambda_1+\lambda_2)\left(1-\frac{\lambda_1}{\mu_1}\right)\left(1-\frac{\lambda_2}{\mu_2}\right)}\right] < \left(\left(1-\frac{\lambda_1}{\mu_1}\right)\left(1-\frac{\lambda_2}{\mu_2}\right)\right)^{-1}.$$

These findings are explained by *coordination idleness* which is incurred when only one resource works in an individual task (while the other individual queue is empty and hence its resource idles) although the collaborative queue has work. As shown in the simulation results in Fig. 6 for the symmetric BC network with $m_i = 0.5$, coordination idleness increases as ρ^{BN} increases and consumes the entire "idleness budget" $1-\rho^{\mathrm{BN}}$ when $\rho^{\mathrm{BN}} = 3-\sqrt{5} \simeq 0.764$.

Nonpreemptive priority completes each collaborative service and thus has smaller coordination idleness than preemptive priority, which explains its smaller capacity loss. Notice that neither policy idles a non-empty individual queue and thus none incurs switching idleness.

[5] With preemption and static priority, indivisibility does not make any difference, so that this result applies also to bandwidth sharing. Indeed, Equation (1) is an instance of Example 1 in Bonald and Massoulié (2001).

[6] One can derive a necessary, but complex, condition for stability using hitting times of a two-MM1-queue system.

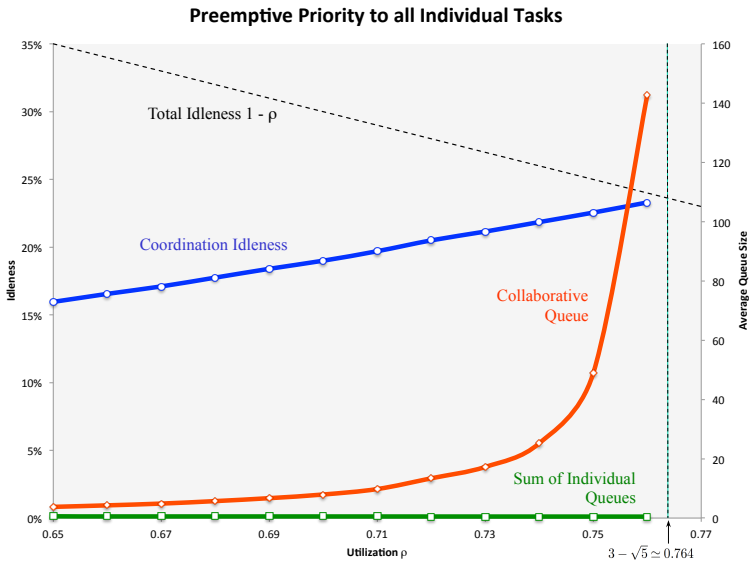

Figure 6 **Coordination idleness under preemptive priority to both individual tasks leads to a capacity loss.**

The essential insight is that misaligning priority and collaboration levels entails a capacity loss. Alternatively, to not lose any capacity, one cannot give highest priority to *both* tasks at the lowest collaboration level (the individual queues). This stark trade-off holds for any stabilizing policy and leads to our main impossibility result:

Theorem 1 (I^2 Impossibility) *Consider the BC network with both resources being bottlenecks (i.e., $1-\rho_i = 1-\rho^{BN}+o(1-\rho^{BN})$) . Then,* **any** *sequence of preemptive policies that does not lose any capacity has at least one large individual queue i, meaning that*

$$\liminf_{\rho^{BN}\uparrow 1}(1-\rho^{BN})\mathbb{E}Q_i^{\rho^{BN}}(\infty) > 0. \tag{3}$$

Any *sequence of non-preemptive policies that does not lose any capacity has two large individual queues, meaning that* (3) *holds for* **both** *individual queues.*

The theorem shows that it is impossible to keep both individual queues small without losing capacity.[7] It also shows there is a difference between preemptive and non-preemptive in the ability to prioritize *one* individual queue (I-Priority): *coordination* idleness prevents the preemptive prioritization of both individual queues without losing capacity but allows for the preemptive prioritization of one individual queue while retaining full capacity. In the proof of the theorem we build a preemptive policy that does exactly that. In contrast, additional *switching idleness* prevents the non-preemptive prioritization of even one individual queue.

[7] The proof that at least one individual queue must be large does not rely on the indivisibility of resources.

3.2. Match: Prioritizing the Collaborative Task

Matching priority levels with collaboration levels means giving highest priority to the collaborative queue, called C-Priority in the BC network. *Preemptive* C-Priority can keep the collaborative queue small without losing any capacity. Indeed, that policy effectively breaks any collaboration constraints and is equivalent to the benchmark (two-class $M/M/1)^2$ network of Fig. 2. Preemptive C-priority incurs no switching or coordination idleness and hence no capacity loss.

Non-preemptive policies that prioritize the collaborative activity do not incur coordination idleness but do suffer from excessive switching idleness incurred by minimizing the collaborative queue. Consider, for example, an arrival to an empty collaborative queue while both resources are working in their individual tasks. If resource 1 completes service first, the policy forces it to idle and wait for resource 2 to complete its service before switching to the collaborative task, even though resource 1 still has work in queue 1. This *switching idleness* increases as ρ^{BN} increases and leads to a capacity loss (which we will quantify in the next subsection).

The essential insight is that matching priority and collaboration levels allows a small collaborative queue and maximal capacity under preemptive service. However, making the collaborative queue small under non-preemption entails a capacity loss. This stark trade-off holds for any stabilizing non-preemptive policy:

Theorem 2 (Matching = C Priority) *Consider the BC network with both resources being bottlenecks (i.e.,* $1-\rho_i^{\rho^{BN}} = 1-\rho^{BN}+o(1-\rho^{BN})$*). Preemptive C-priority keeps the collaborative queue small without losing any capacity:*

$$\mathbb{E}Q_0^{\rho^{BN}}(\infty) = \frac{\lambda_0}{\mu_0-\lambda_0} \Rightarrow \lim_{\rho^{BN}\uparrow 1}(1-\rho^{BN})\mathbb{E}Q_0^{\rho^{BN}}(\infty) = 0. \tag{4}$$

In contrast, **any** *sequence of non-preemptive policies that does not lose any capacity has a large collaborative queue:*

$$\liminf_{\rho^{BN}\uparrow 1}(1-\rho^{BN})\mathbb{E}Q_0^{\rho^{BN}}(\infty) > 0. \tag{5}$$

Given that the capacity loss under nonpreemptive C-priority stems from excessive switching idleness, we next discuss two policies that control the switching and hence reduce, if not eliminate, the capacity loss.

3.3. Trading off Priority and Capacity: Collaborative Threshold Priority

To minimize switching idleness it is natural to consider a polling policy that serves the collaborative queue until exhaustion (i.e., until $Q_0 = 0$), after which resources switch to their individual tasks and serve those until all individual queues are empty before moving back to the collaborative queue.

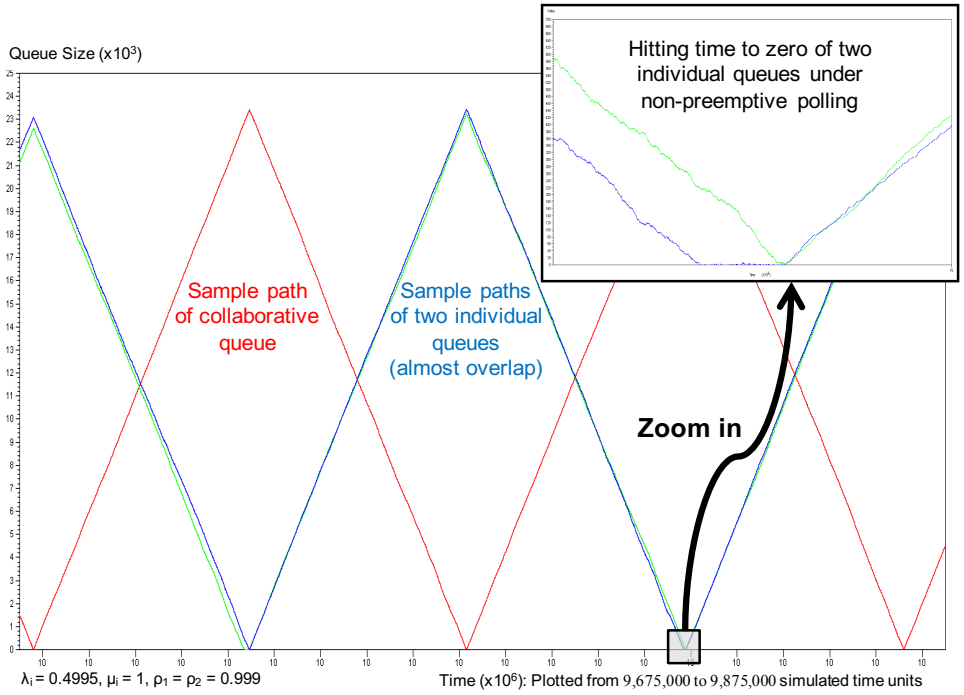

Figure 7 **Polling leads to extreme queue oscillations and hence large expected queue sizes in the BC network.**

In addition to minimizing switching idleness, polling incurs no capacity loss and is a decentralized policy that also allows a natural human discretion: people switch tasks only when their current task is exhausted. Polling is "democratic" by not favoring any specific queue. This, however, leads to extreme queue oscillations (Fig. 7). The expected sum of queue sizes is finite but very large and grows much faster than in the reference M/M/1 system where queues scale at rate $(1-\rho^{\mathrm{BN}})^{-1}$:

Proposition 2 (Polling) *Non-preemptive polling incurs no capacity loss and stabilizes the BC network for any $\rho^{BN}<1$ yet has very large queues: If both resources are bottlenecks (i.e., $1-\rho_i^{\rho^{BN}} = 1-\rho^{BN}+o(1-\rho^{BN})$), then queues scale super-linearly in $(1-\rho^{BN})^{-1}$:*

$$\liminf_{\rho^{BN}\uparrow 1}(1-\rho^{BN})^{3/2}\mathbb{E}Q_+^{\rho^{BN}}(\infty)>0.$$

The culprit behind the superlinear scaling is coordination idleness stemming from the requirement that *both* individual queues be empty before moving back to the collaborative queue (Fig. 8). The total time the individual queues are served during one cycle is the *largest* hitting time of zero

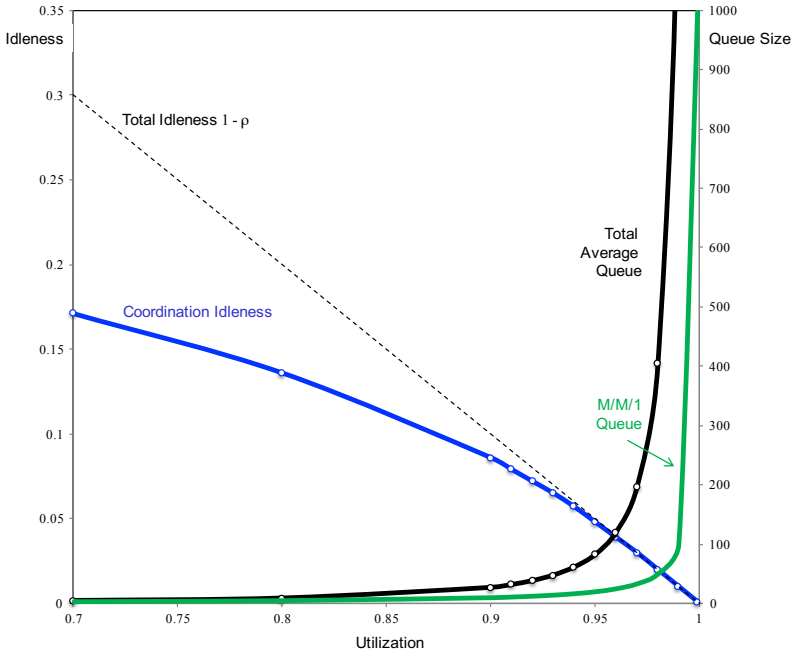

Figure 8 Coordination idleness in the BC network under polling yields much larger queues than in M/M/1 queue.

among all individual queues (Fig. 7). There is no switching idleness here as moving to the collaborative task happens only when all individual queues are empty.[8] Thus, by eliminating switching idleness, polling incurs excess coordination idleness that retains stability yet leads to significant queues. In essence, polling swaps priority control for capacity control.

We seek an intermediate solution that trades off priority and capacity by *partially* matching priority and collaboration levels. Consider the C-priority policy with added threshold S: When resources are serving the individual queues, an alarm sounds when the collaborative queue hits level S. Resources move to the collaborative queue as soon as possible (those that are idle immediately, the others upon completion of their current service) and serve the collaborative queue to exhaustion before moving back to the individual queues.

This "collaborative threshold priority" policy retains the benefits of polling by avoiding excessive switching yet bounds the switching idleness per cycle by the "switching time" T^s, which is the time from the alarm sounding until all resources have moved to the collaborative task. With iid exponential service times, T^s is the maximum of 2 service times. More generally, if there were J individual queues, the expected switching time is

$$m_1 \log(1+J) < \mathbb{E}T^s = m_1 \sum_{j=1}^{J} \frac{1}{j} < m_1(1+\log J), \tag{6}$$

[8] A similar phenomenon arises in the communication network studied in Simatos et al. (2014) where a protocol that maximizes throughput leads to super-linear growth of the queues due a similar polling behavior; notice the similarity of their Figure 1 and our Figure 7.

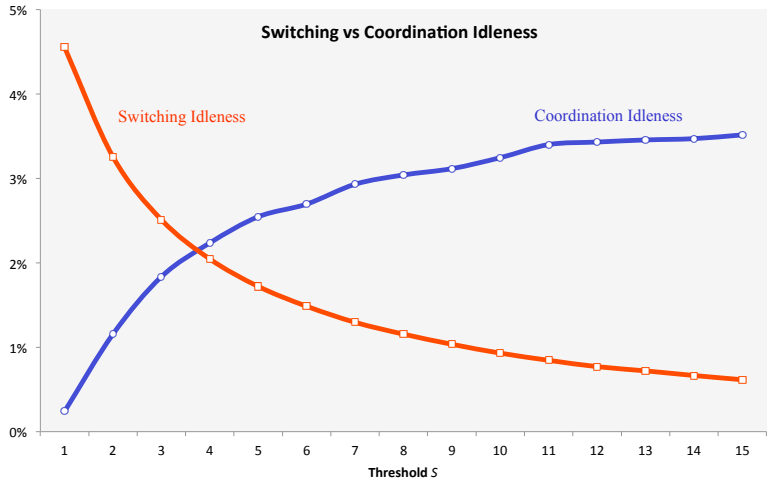

Figure 9 The collaborative threshold priority policy trades off switching and coordination idleness (symmetric BC network with $m_i = 0.5$ and $\lambda_i = 0.95$ so $\rho_j = 0.95$).

so that the expected switching idleness per cycle per resource $\mathbb{E}T^s - m_1$ is bounded by $m_1 \log J$. Of course, T^s depends on the policy: If we require a predetermined sequence in which resources stop and move to the collaborative queue (e.g., cheapest resources idles first), then T^s is the sum of J service times and $\mathbb{E}T^s \leq m_1 J$.

For any sequencing rule, we can divide the switching idleness by the length of a cycle (see companion) to yield the average switching idleness rate for resource j:

$$\frac{\lambda_j m_j (\mathbb{E}T^s - m_j)}{\frac{S}{\lambda_0} + m_j}. \tag{7}$$

A smaller threshold S yields a smaller collaborative queue but requires more switching which reduces capacity. This yields an explicit stability condition that quantifies the capacity loss:

Proposition 3 *The BC network is stable under the non-preemptive Collaborative Threshold Priority policy provided that for each resource $j \in \{1, 2\}$:*

$$\rho_j + \frac{\lambda_j m_j (\mathbb{E}T^s - m_j)}{\frac{S}{\lambda_0} + m_j} < 1.$$

Moreover, there exists k such that with $S = k(1-\rho^{BN})^{-1}$ the policy incurs no capacity loss (maximizes throughput) and total queue count scales optimally: $\limsup_{\rho^{BN} \uparrow 1} (1-\rho^{BN}) \mathbb{E} Q_+^{\rho^{BN}}(\infty) < \infty$.

Remark: Note that the Collaborative Threshold Priority policy with $S = 1$ reduces to the (static) C-priority policy so that Proposition 3 quantifies its capacity loss, as foreshadowed earlier.

The proposition reflects the fact that, while S determines the average collaborative queue, the ratio λ_0/S determines the switching frequency. As ρ^{BN} increases we must reduce the switching

frequency to preserve stability. The consequent increase of S *trades off switching and coordination idleness* as shown in Figure 9. For small values of S, the servers switch too frequently and hence switching idleness dominates. For large values of S switching is less frequent but coordination idleness dominates due to the increase in instances where one individual queue is empty but the resources are not working on the collaborative task although it has a queue.

The proposition is consistent with the impossibility result: A lower threshold S gives higher non-preemptive priority to the collaborative task but incurs higher switching idleness and capacity loss. Eliminating the capacity loss requires a high threshold and yields a higher collaborative queue.

The remainder of this paper will show that the benefits of aligning task priorities and collaboration levels extend to parallel networks. These more general networks also serve as a basis for a general definition of switching and coordination idleness.

4. Parallel network model and collaboration architecture

Parallel networks generalize the BC network: they are single stage networks in which customers arrive into a queue, are served, and then leave the system. The lack of inter-queue routing in parallel networks facilitates the analysis of the impact of collaboration and multitasking.

There is a set $\mathcal{K} = \{1, \ldots, K\}$ of resources and a set $\mathcal{J} = \{1, \ldots, J\}$ of activities. Each activity is associated with a single buffer: there are J buffers. The average arrival rate into buffer i is $\lambda_i > 0$. The average processing time of activity i is $m_i > 0$. Arrivals follow independent Poisson processes and service times are independent across customers and exponentially distributed The $K \times J$ *resource-activity incidence matrix* A has $A_{ki} = 1$ if resource k is required for activity i, and $A_{ki} = 0$ otherwise. The distinguishing feature of collaborative networks is that A has at least one column (activity) with multiple 1's (collaborative resources). The distinguishing feature of multitasking is that at least one row (resource) has multiple 1's. For $i \in \mathcal{J}$, we let $\mathcal{R}(\{i\})$ be the set of resources required for activity i (i.e., $k \in \mathcal{R}(\{i\})$ if $A_{ki} = 1$). More generally, $\mathcal{R}(\mathcal{A})$ is the set of resources required for some activity in the set $\mathcal{A} \subseteq \mathcal{J}$: $k \in \mathcal{R}(\mathcal{A})$ if $k \in \mathcal{R}(\{i\})$ for some $i \in \mathcal{A}$. To avoid trivialities we assume that each resource is required for at least one activity so that $\mathcal{R}(\mathcal{J}) = \mathcal{K}$. We let $\mathcal{S}(i, j)$ be the set of resources shared by activities i and j: $\mathcal{S}(i, j) = \mathcal{R}(\{i\}) \cap \mathcal{R}(\{j\})$.

Parallel networks, as shown in Figure 10, allow us to isolate the effect of collaboration and multitasking yet are still surprisingly subtle. The only dependence between activities is through possibly overlapping resources and there are two extremes: If each activity i uses a dedicated resource i, then we have J independent single-class single-server systems. In contrast, if all activities share the same resources $\mathcal{R}(1) = \mathcal{R}(2) = \ldots = \mathcal{R}(J)$ we recover a multiclass $M/M/1$ queuing system. In general, collaboration and multitasking fall between these two extremes: some activities share some resources ($\mathcal{R}(i)$ and $\mathcal{R}(j)$ may overlap) and some resources process multiple activities.

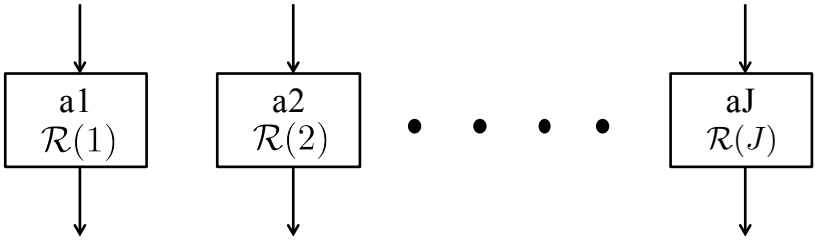

Figure 10 **Parallel networks are single stage networks featuring collaborating and multitasking resources.**

We adopt the following notation: All vectors are represented as column vectors. The vector of all ones is denoted by e, or by e_d to emphasize the dimension d. Componentwise multiplication is denoted by $*$ so that $v * w$ represents the vector with i-th component $v_i w_i$. The component sum $\sum_k v_i$ is denoted by v_+.

Remember the traditional notions of utilization and capacity: the average workload arriving into each queue per unit of time is $m * \lambda$, which we call the activity load vector ρ^a. Summing over the relevant activities yields the resource utilization vector ρ:

$$\rho^a(\lambda) = m * \lambda \text{ and } \rho(\lambda) = A\rho^a(\lambda). \tag{8}$$

The bottleneck resources are those with the highest utilization, denoted by $\rho^{\mathrm{BN}}(\lambda) = \max_{k \in \mathcal{K}} \rho_k(\lambda)$. It is useful to formulate this as a linear program, called the static planning problem (SPP):

$$\rho^{\mathrm{BN}}(\lambda) = \min_{\rho \in \mathbb{R}_+} \quad \{\rho \ : \ A\rho^a(\lambda) \leq \rho e\}\,. \tag{SPP}$$

If $\rho^{\mathrm{BN}}(\lambda) \leq 1$, then ρ^{BN} is the fraction of time the bottlenecks are busy processing. The set of SPP-feasible activity load vectors $x = m * \lambda$ is the polyhedron

$$\mathcal{P} = \{x \in \mathbb{R}_+^J : Ax \leq e\}. \tag{9}$$

Traditional networks can typically handle arrival rates λ in the interior of $\mathcal{P}$ (i.e., with $\rho^{\mathrm{BN}}(\lambda) < 1$): there is a control policy that makes the network stable with queues that remain finite in expectation. With collaboration and multitasking, this is not necessarily true because resources do not work in isolation.

Rather than merely requiring that each individual resource is not overloaded, we must require that the network as a whole is not overloaded while preventing any processing conflicts. The resource-activity matrix A captures these processing conflicts—activities that cannot be performed simultaneously because they share resources. Based on these we can construct configuration vectors which specify which activities are being executed: A feasible configuration vector v is a binary non-zero J-vector such that $v_i = v_j = 1$ if activities i and j do not share resources ($\mathcal{S}(i,j) = \emptyset$) and

can thus be performed simultaneously. The number of feasible configurations is at least as large as the number of activities J. Indeed, each activity can be performed in isolation which means that the unit vectors in $\mathbb{R}^J$ are natural feasible configuration vectors. In the absence of collaboration and multitasking, all activities can be performed simultaneously and all 2^J binary J-vectors are feasible configurations.

Notice that a binary vector v is a feasible configuration if and only if $\sum_i A_{ki} v_i \leq 1$ for each resource k. In particular, the feasible configuration vectors are the *integer* vertices of the polyhedron $\mathcal{P}$. Let C be the matrix of these integer vertices; i.e., each column of the matrix C is a feasible configuration vector. (Of course, $AC \leq e_K e'_J$.) With some abuse of notation we write $a \in C$ when a is a column of C.

To keep up with the average workload $m * \lambda$ arriving into each queue per unit of time, the corresponding average amount of time the network must be processing per unit of time—the network utilization— is given by the solution to the following linear program

$$\begin{aligned} \rho^{\text{net}}(\lambda) = \min_{\pi \in \mathbb{R}^J_+} \ & e'\pi \\ \text{s.t.} \ & C\pi = m * \lambda, \end{aligned} \tag{SPPC}$$

where π_c is interpreted as the long-run average allocation to configuration c. Denote by $\bar{\pi}$ the optimal time allocation in this Static Planning Program for Collaboration (SPPC). The *network capacity* is the set of throughput vectors $\lambda \geq 0$ for which $\rho^{\text{net}}(\lambda) = 1$, which implies that the network is fully utilized.

Notice that a feasible activity load vector $m * \lambda$ is a convex combination of the configuration vectors and thus belongs to $\mathcal{P}$ (which also can have non-integer vertices). Hence, $\rho^{\text{net}} \geq \rho^{\text{BN}}$ meaning that the network must work longer than the bottleneck resources (which may not be able to work in parallel because of processing conflicts). In GVM, we called this gap:

$$\text{Unavoidable Bottleneck Idleness UBI}(\lambda) = \rho^{\text{net}}(\lambda) - \rho^{\text{BN}}(\lambda) \geq 0. \tag{10}$$

Theorem 4.2. in GVM states that $\rho^{\text{net}}(\lambda) = \rho^{\text{BN}}(\lambda)$ and $\text{UBI}(\lambda) = 0$ for any parameters m, λ, or any other probabilistic assumptions, if and only if the polyhedron $\mathcal{P}$ is integral. Indeed, only then do the configuration vectors span the feasible region of the (SPP). Notice that the essential condition involves matrix A and is independent of other parameters. To gain structural insight, we presented a graph representation of the matrix A as a formal tool to characterize properties of what we call the *collaboration architecture*. The graph has a node for each activity and an edge between two nodes if their resource sets overlap. Our main result in GVM (Theorem 4.4) was that a "nested" collaboration architecture features no UBI. We refer the reader to GVM for the formal (general) definition of nestedness.

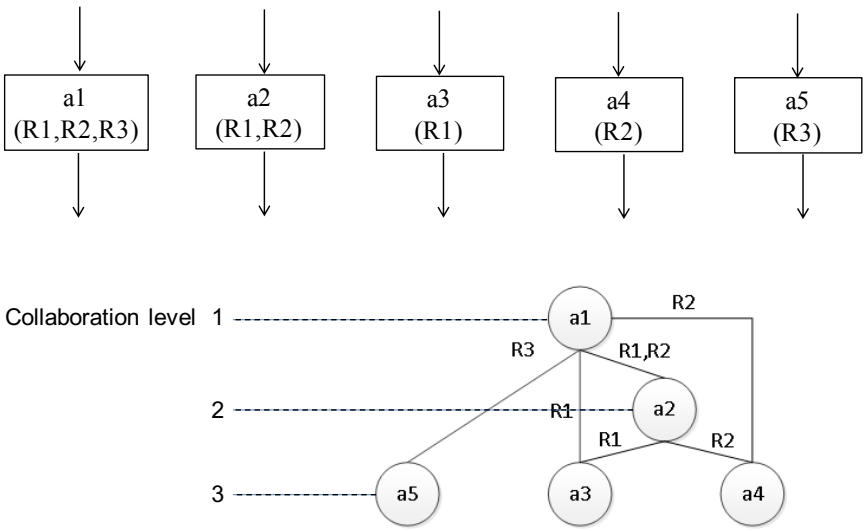

Figure 11 **A network with hierarchical collaboration architecture (top) and its collaboration graph (bottom).**

Hierarchical architectures are a special subset of nested architectures that are most relevant to this paper. A hierarchical architecture can be represented by a graph where nodes are arranged in "collaboration levels" $i = 1, \ldots, d$. Two nodes at the same level do not share resources and, similar to priority levels, the top collaboration level 1 contains activities that require the highest number of collaborating resources. Figure 11 shows a simple example while GVM presents an algorithm for constructing such a leveled graph from the adjacency matrix A. Notice that in Figure 11 activities 3 and 4 do not share resources. Activities 1 and 3 share resource 1 and so does activity 2 that is on a path from a3 to a1. Similarly, a1 and a4 share resource 2 which is also used by a2 that is on a path from a4 to a1. Hierarchical architectures have collaboration graphs where each path from a top node to a bottom node has at least one resource shared by all activities on that path.

5. Dynamics of network workload $W^{\mathbf{net}}$

A control rule specifies which configuration vector is active at each time t. Let $T(t)$ be the cumulative allocation (vector) process. Its component $T_c(t)$ is the cumulative amount of time a configuration vector c is active during $[0, t]$:

$$T_c(t) = \int_0^t \mathbb{1}\Big\{\text{configuration vector } c \text{ is active at time } \tau\Big\} d\tau,$$

where $\mathbb{1}\{\cdot\}$ denotes the indicator function. Given that only one configuration can be active at any point in time, the total time the network is processing during $[0, t]$ is $\sum_c T_c(t) = e'T(t) \leq t$. The total time the network is idle (and no configuration is active) during $[0, t]$ is $I^{\text{net}}(t) = t - e'T(t)$.

To translate queue workload into network workload and activity load into network utilization we first abstract from detailed dynamics and consider a long-run average time allocation vector π and use the configuration matrix C. For a constant time-allocation vector π, the total amount of

time activity i is processed equals $(C\pi)_i$. We define the *network workload* $W^{\rm net}(Q)$ as the minimal expected time needed to process the work embodied in the queue length vector Q:

$$\begin{aligned} W^{\rm net}(Q) = \min_{\pi\in\mathbb{R}^J_+} \ & e'\pi \\ \text{s.t.} \ & C\pi \geq m*Q. \end{aligned} \tag{11}$$

For the BC and BC+ networks of Figure 1 this yields the intuitive expressions:

$$W^{\rm net}_{\rm BC}(Q) = m_1Q_1 + \max\{m_2Q_2, m_3Q_3\} \ \text{ and } \ W^{\rm net}_{\rm BC+}(Q) = m_1Q_1 + m_2Q_2 + m_3Q_3. \tag{12}$$

It is instructive to consider the dual linear program to (11):

$$\begin{aligned} \max_{y\in\mathbb{R}^{\mathbf{I}}_+} \ & y'(m*Q) \\ \text{s.t.} \ & y'C \leq e'. \end{aligned} \tag{13}$$

Letting $y^\star(Q)$ denote the optimal solution, strong duality then gives us a simple decomposition of network workload directly in terms of queue workload:

$$W^{\rm net}(Q) = \sum_j y^\star_j(Q) m_j Q_j. \tag{14}$$

There are two extreme deconstructions depending on A or C: When A is the identity matrix (networks without collaboration or multitasking), then $W^{\rm net}(Q) = \max_j(m_jQ_j)$ and $y^\star$ is a piecewise linear function of Q. When C is the identity matrix (as in the BC+ networks), then $W^{\rm net}(Q) = \sum_j(m_jQ_j)$ and $y^\star = e$ is the constant vector of ones, indicating that resources perform as a single resource. In general, the dual variables of the instantaneous network workload depend on Q.

A long run average view simplifies these dual variables. Consider a network with initial queue vector $Q(0)$ and arrival and service processes $A(t)$ and $S(t)$, which denote the number of arrivals and service completions, respectively, during $[0,t]$. Recall that $T_c(t)$ is the cumulative amount of time configuration vector c is active during $[0,t]$ so that:

$$Q(t) = Q(0) + A(t) - S\circ CT(t), \tag{15}$$

where $S\circ CT(t)$ is the vector with components $S_j((CT)_j(t))$. Dividing both sides by t, letting $t\to\infty$ and recalling the strong law that $A_j(t)/t \to \lambda_j$ and $S_j(t)/t \to 1/m_j$, then in a stable network, we must have

$$C\pi = m*\lambda \ \text{ and } \ \pi = \lim_{t\to\infty}\frac{T(t)}{t}. \tag{16}$$

As shown earlier, the optimal time-allocation rates $\bar\pi$ satisfy (SPPC). If $\bar\pi_k > 0$, we say that configuration vector k is optimal; otherwise it is a suboptimal configuration. We label the configurations to allow the following block partitioning

$$\bar\pi = [\pi^\star, 0] \text{ and } C = [C^\star, C^0], \tag{17}$$

where $\pi^\star > 0$ and $C^\star$ contains all optimal configuration vectors.

Let $\bar{y} = y^\star(\lambda)$ denote the solution to (13) when $Q = \lambda$ (this is the dual of SPP). By strong duality $\sum_j \bar{y}_j m_j \lambda_j = \rho^{\text{net}}$ and complementary slackness further yields that $\bar{y}' C_{\cdot,k} = 1$ if the k^{th} column is an optimal configuration, meaning $\bar{\pi}_k > 0$. In matrix notation, recalling (17), we have

$$\bar{y}' C^\star = e'. \tag{18}$$

Let the 0 configuration mean that no resource is processing (the network idles). Then,

$$\bar{y}' C \bar{\pi} = \bar{y}' C^\star \pi^\star = e' \pi^\star = 1 - \bar{\pi}_0, \tag{19}$$

where $\bar{\pi}_0$ is the optimal idleness rate (the fraction of time the zero configuration is used). The dual variables $\bar{y}$ provide us with a simplification: it can be shown that in heavy traffic as $\rho^{\text{net}} \uparrow 1$, then

$$W^{\text{net}}(Q^{\rho^{\text{net}}}(t)) = \sum_j y_j^\star(Q^{\rho^{\text{net}}}(t)) m_j Q_j^{\rho^{\text{net}}}(t) \approx \sum_j \bar{y}_j m_j Q_j^{\rho^{\text{net}}}(t).$$

Henceforth, the network workload process refers to

$$W^{\text{net}}(t) = \sum_j \bar{y}_j m_j Q_j(t). \tag{20}$$

Essentially, $W^{\text{net}}(t)$ is the expected time the network needs to process the queue vector $Q(t)$ while using the steady-state optimal time-allocations $\bar{\pi}$. Combining (15) and (20) we have that

$$\begin{aligned} W^{\text{net}}(t) &= W^{\text{net}}(0) + \sum_j \bar{y}_j m_j A_j(t) - \sum_j \bar{y}_j m_j S_j((CT)_j(t)) \\ &= W^{\text{net}}(0) + \sum_j \bar{y}_j m_j \lambda_j t - \bar{y}' C T(t) + M(t), \end{aligned}$$

where M is the deviation from the fluid (first moment) processes:

$$M(t) = \sum_j \bar{y}_j m_j \Big(A_j(t) - \lambda_j t \Big) + \sum_j \bar{y}_j m_j \Big(\mu_j (CT)_j(t) - S_j((CT)_j(t) \Big).$$

Similar to (17), block partition the control vector as

$$T(t) = [T^\star(t); T^0(t)] \tag{21}$$

where $T^\star$ denotes the allocation to all optimal configurations and T^0 to the suboptimal configurations. Complementary slackness then yields

$$\bar{y}' C T(t) = e' T^\star(t) + \bar{y}' C^0 T^0(t).$$

Recall that $e' T(t) = t - I^{\text{net}}(t)$ so that

$$\bar{y}' C T(t) = t - I^{\text{net}}(t) - (e' - \bar{y}' C^0) T^0(t).$$

Strong duality yields that $\sum_j \bar{y}_j m_j \lambda_j = \rho^{\text{net}}$ so that we arrive at

$$W^{\text{net}}(t) = W^{\text{net}}(0) - (1-\rho^{\text{net}})t + (e' - \bar{y}'C^0)T^0(t) + M(t) + I^{\text{net}}(t). \tag{22}$$

This is the first indication that usage of suboptimal configurations increases the network workload, which will be further analyzed in the next section. Conversely, for networks that have no suboptimal configurations (as in the BC+ network where $C = C^\star$ is the identity matrix) the "damaging term" $(e' - \bar{y}'C^0)T^0(t) = 0$ and equation (22) reduces to the single server equation. In general, equation (22), together with the fact that $e' - \bar{y}'C^0 \geq 0$, allows us to state the following result.

Theorem 3 *If $\rho^{net} = \rho^{BN+}$ $UBI > 1$ then the network is transient under any policy, i.e.,*

$$\liminf_{t\to\infty} \frac{W^{net}(t)}{t} > 0.$$

6. Formal Definitions of Coordination and Switching Idleness

We have just established that "inefficiency" is captured by the suboptimal-configurations term $(e' - \bar{y}'C^0)T^0(t)$. Suboptimal configurations may be used for several reasons: when resources have conflicting priorities (coordination idleness), when we cannot preempt service to switch to an optimal configuration (switching idleness) or when the only available work is in suboptimal configurations (availability idleness).

Switching idleness is incurred when activating a resource—putting an idle resource to work in an optimal configuration—does not necessitate the movement of any other resource. This happens when there is an available optimal configuration that is a superset of the currently used configuration. Consider, for example, the BC network (Fig. 1) with a non-preemptive priority to the collaborative task. Suppose that $Q_0 = 0, Q_1 > 0, Q_2 > 0$ and we witness an arrival to the empty collaborative queue while both resources are working in their individual tasks (the configuration is $(0,1,1)$). If resource 1 completes service first, it must wait for resource 2 to complete its service before switching to the collaborative task, even though resource 1 still has work in queue 1: we use suboptimal configuration $(0,0,1)$ while the optimal configuration $(0,1,1)$ is still available. Hence, the term *switching idleness*.

In what follows, given $q \in \mathbb{R}_+^J$ and a configuration vector b, we write q_b for the sub-vector that has q_i for i such that $b_i = 1$. Thus, $Q_b(t)$ is the vector of queues corresponding to activities that are active under configuration b.

Definition 1 (Switching Idleness) *For suboptimal configuration $a_k = C^0_{\cdot k}$ let*

$$I^s_k(t) = \int_0^t \mathbb{1}\{S^*(a, Q(t)) \neq \emptyset\} dT^0_k(t),$$

where $S^*(a,q) := \{b \in C^* : b > a, q_b > 0\}$. The network's switching idleness is given by $I^s_+(t) = (e' - \bar{y}'C^0)I^s(t)$.

Coordination idleness is incurred when putting an idle resource to work *necessitates* moving other resources. When we use a suboptimal configuration a while there exists an optimal configuration b that would utilize more resources but would require moving a resource from an activity it is currently processing. Consider, for example, the BC network (Fig. 1) at a time when $Q_0 > 0, Q_1 = 0, Q_2 > 0$. If we prioritize the individual activities, then we would use suboptimal configuration $a = (0,0,1)$ where only resource 2 is processing while the optimal configuration $b = (1,0,0)$ would utilize both resources. Yet that configuration would require moving resource 2 from activity 2 that it is currently processing. In that case, resource 1 is incurring coordination idleness. Using the exclusive-or[9] operator $\otimes$, this means that for these two configurations a and b, $b \otimes a \neq 0$ and neither $b > a$ nor $a > b$. We can now define:

Definition 2 (Coordination Idleness) *For suboptimal configuration $a_k = C^0_{\cdot k}$, let*

$$I^c_k(t) = \int_0^t \mathbb{1}\{S^*(a_k, Q(t)) = \emptyset \text{ and } X^*(a_k, Q(t)) \neq \emptyset\} dT^0_k(s), \tag{23}$$

where

$$X^*(a_k, q) := \{b \in C^* : b \otimes a_k \neq 0, q_b > 0 \text{ and neither } b > a_k \text{ nor } a_k > b\}.$$

The network's coordination idleness is given by $I^c_+(t) = (e' - \bar{y}'C^0)I^c(t)$.

The indicator sets for coordination and switching idleness are disjoint so that $I^c_k(t) + I^s_k(t) \leq T^0_k(t)$. Availability idleness refers to the remaining time that the suboptimal configuration k is used while not incurring coordination or switching idleness:

Definition 3 (Availability Idleness) *For suboptimal configuration $a_k = C^0_{\cdot k}$ let*

$$I^a_k(t) := T^0_k(t) - I^c_k(t) - I^s_k(t). \tag{24}$$

The network's availability idleness is given by $I^a_+(t) = (e' - \bar{y}'C^0)I^a(t)$.

[9] For two binary vectors a and b, $a \otimes b = a + b$ modulo 2.

Availability idleness is the cumulative time that suboptimal configuration a_k is used while there is no optimal configuration available that is not a subset of a_k. In the symmetric BC network, $I_k^a(t)$ increases when only one individual queue has work but the other two queues are empty. Such events are sufficiently rare that availability idleness does not amount to a capacity loss in the BC network or more generally:

Theorem 4 *Consider a parallel network with a hierarchical*[10] *collaboration architecture and a policy that, when the only configurations available are sub-optimal, prefers a configuration that utilizes a bottleneck resource. Then, given $\bar{y} \in \mathcal{Y}$, it holds that, almost surely,*

$$\limsup_{t\to\infty} \frac{1}{t} I_+^a(t) \leq 1 - \rho^{BN}.$$

In contrast to availability idleness, coordination and switching idleness do lead to capacity losses as we have shown for the BC network. Theorem 4 is easy to verify in the symmetric BC network: $\lambda_1 m_1 = \lambda_2 m_2$ and $(1, 1/2, 1/2)$ is an optimal $\bar{y}$. The time that the suboptimal configuration $(0, 1, 0)$ is used while no other optimal configuration ($(1, 0, 0)$ or $(0, 1, 1)$) is available is bounded from above by the amount of time that resource 2 has no work anywhere in the network. This is bounded by the time that the resource would have no work if it could work in isolation, which in turn is bounded by $1 - \rho^{\mathrm{BN}}$.

Denoting $I^{\mathrm{UBI}}(t) = (\rho^{\mathrm{net}} - \rho^{\mathrm{BN}})t$, we can capture all four types of collaboration idleness in (22):

$$\begin{aligned} W^{\mathrm{net}}(t) = W^{\mathrm{net}}(0) &- (1 - \rho^{\mathrm{BN}})t + M(t) \\ &+ I^{\mathrm{UBI}}(t) + I_+^s(t) + I_+^c(t) + I_+^a(t) \\ &+ I^{\mathrm{net}}(t). \end{aligned} \tag{25}$$

7. Possibilities for Parallel Networks

This section extends the main results of the BC network to parallel networks.

We introduce *hierarchical priority policies* that match priority levels to collaboration levels as follows: At any time t, resource k is assigned, amongst its activities with positive queues, to the activity at the highest collaboration level. In particular, a resource will work in level l only if the queues of all its activities at higher levels $1, \ldots, l-1$ are empty. In the BC network, for example, the hierarchical priority policy reduces to static priority to the collaborative activity and its preemptive version has all desirable properties (Theorem 2) that extend to parallel networks:

[10] The proof applies to the larger class of "nested" collaboration architectures.

Theorem 5 (Matching–Hierarchical Preemptive Priorities) *Consider a parallel network with hierarchical collaboration architecture. A* **preemptive** *hierarchical priority policy that matches task priorities with collaboration levels keeps the asymptotic workload only at the activities at the lowest collaboration level l and achieves optimal scaling without losing any capacity:*

$$\limsup_{\rho^{BN}\uparrow 1}(1-\rho^{BN})\mathbb{E}Q_{+}^{\rho^{BN}}(\infty)<\infty \text{ and } \limsup_{\rho^{BN}\uparrow 1}(1-\rho^{BN})\mathbb{E}Q_{i<l}^{\rho^{BN}}(\infty)=0.$$

Non-preemptive service performs fundamentally differently from preemptive service in networks where indivisible resources collaborate, even in heavy-traffic. Non-preemptive hierarchical policies incur excessive switching idleness that entails a capacity loss. To reduce the switching idleness and capacity loss, we introduce *Collaborative Hierarchical Threshold Priority* policies which generalize the collaborative threshold priority policy of the BC network to parallel networks with hierarchical collaboration architectures. We write $i \lesssim j$ if there is a path in the collaboration graph between i and j and i is at a lower level than j. Then define the collaborative hierarchical priority policy with vector of thresholds $\mathbf{S}$ as:

(i) Upon completing service in queue i, a resource moves to the highest collaboration level activity j for which $i \lesssim j$ and $Q_j \geq S_j$, and serves that queue.

(ii) If no queue is found in (i), continue serving the current queue i if it is non-empty.

(iii) If both (i) and (ii) fail, serve the highest-level activity $j \lesssim i$ that has a nonempty queue.

(iv) If all (i)-(iii) fail, the resource waits until one of the above conditions holds.

For example, resource R2 in Fig. 11 prioritizes a1 over a2 and a2 over a4.

Of course, whether or not the resource can work in a queue to which it moves depends on other resources as well. Notice that this policy prescribes actions to resources rather than to the network as a whole. This "local policy" does not introduce conflicts due to the hierarchical collaboration architecture which guarantees that when a resource k moves to an activity j so does, upon service completion, any other resource i required for that activity.

One may worry that too much switching is being introduced because some collaborative queues may be left before they are exhausted. Yet that worry is unsubstantiated because we can bound the switching idleness. Effectively, the highest level queues will be served to exhaustion before moving to lower level queues. That means that for thresholds that grow as $(1-\rho^{\mathrm{BN}})^{-1}$ it will take roughly the same amount of time $\mathcal{O}((1-\rho^{\mathrm{BN}})^{-1})$ until switching. That guarantees that the switching idleness is at most $\mathcal{O}(1-\rho^{\mathrm{BN}})$, which is the foundation of our last theorem on matching hierarchical priority levels with collaboration levels. An activity j is said to be a leaf node if there is no activity $i \neq j$ such that $i \lesssim j$. Activities $a3, a4$ and $a5$ in Figure 11 are leaf nodes.

Theorem 6 (Matching–Hierarchical Threshold Priorities) *Consider a parallel network with a hierarchical collaboration architecture. A hierarchical threshold priority policy that matches task priorities with collaboration levels and has sufficiently high thresholds controls all non-leaf queues and achieves optimal scaling without losing any capacity: There exist k_j such that with $S_j = k_j(1-\rho^{BN})^{-1}$ the policy maximizes throughput and achieves optimal queue scaling:*

$$\limsup_{\rho^{BN}\uparrow 1}(1-\rho^{BN})\mathbb{E}Q_+^{\rho^{BN}}(\infty) < \infty, \tag{26}$$

while controlling the queue at each non-leaf node j,

$$\limsup_{\rho^{BN}\uparrow 1}(1-\rho^{BN})\mathbb{E}Q_j^{\rho^{BN}}(\infty) \leq k_j. \tag{27}$$

8. Concluding Remarks

Networks with simultaneous collaboration by multiple types of multitasking human or indivisible resources present challenges to capacity management and task prioritization. We introduced and formally defined two types of idleness—coordination and switching—to explain and quantify these challenges. Different task priority policies introduce different combinations and magnitudes of coordination and switching idleness; recall Fig. 4. Both coordination and switching idleness can destabilize the network and lead to capacity losses but can also have more subtle impact (e.g., as under polling).

Our key advice is to match task priorities with the collaboration levels defined by the network's collaboration architecture. Preemptive matched priorities allow collaborative queue control in parallel networks without capacity losses. Non-preemptive policies, however, must trade-off prioritization and capacity, which can be achieved with a hierarchical threshold priority policy. This decentralized policy balances switching and coordination idleness: higher thresholds reduce switching idleness and capacity losses but increase coordination idleness and queue sizes. This explicitly captures the stark tradeoff between capacity and queue control shown in Fig. 12.

Hierarchical architectures are central to our results: They facilitate the decentralized coordination of resources to keep collaborative queues under control (through thresholds) without significantly compromising the total network performance. Aligning task priorities with collaboration levels has consequences for organizational and network design. Our results suggest that only in suitably defined hierarchical organizational structures, collaboration and multitasking can happen without capacity losses. (This may be one benefit of the strong hierarchy in hospitals or complex organizations like the military that require lots of collaboration.) If the desired priorities do

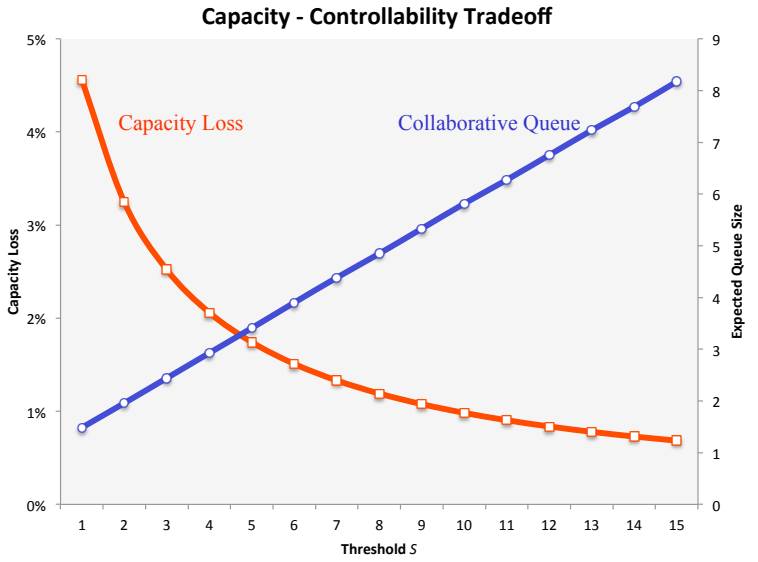

Figure 12 **The Capacity-Controllability Tradeoff for the BC network with a threshold priority policy.**

not match collaboration levels there are two options: one either must install extra buffer capacity or redesign the process (e.g., perhaps by duplicating highly-collaborative activities into more individual activities if possible).

The optimal control of networks with collaboration and multitasking indivisible resources remains an interesting open challenge. One would want to characterize the structure of optimal policies that minimize specific delay costs. Our results contribute to this future pursuit by highlighting the new limitations that these intriguing networks add relative to traditional queuing networks. Of importance for dynamic control in heavy-traffic are the facts that:

(1) Preemption and non-preemption are fundamentally different, even in heavy traffic. Certain prioritizations are impossible under either. But what is achievable under preemptive policies may not be achievable, not even asymptotically, under non-preemptive policies.

(2) Non-negligible thresholds of order $\mathcal{O}(1/(1-\rho^{\mathrm{BN}}))$ are necessary to limit the switching idleness. This is similar to the work on queues with exogenous switchover times (Reiman and Wein (1998)). Switchover times here are, however, endogenous which makes control more difficult.

(3) Hierarchical architectures simplify resource coordination and give simple answers when the costs are aligned with the hierarchy defined by the levels in the collaboration tree. For example, the preemptive hierarchical threshold priority policy may minimize linear holding costs if activities at higher levels in the tree incur higher cost.

References

Bonald, T., L. Massoulié. 2001. Impact of fairness on internet performance. *SIGMETRICS Performance Evaluation Review* **29**(1) 82–91.

Borst, S.C. 1996. *Polling Systems*. CWI.

Csörgo, M., L. Horváth. 1996. *Weighted approximations in probability and statistics*. John Wiley & Sons; Chicester.

Gamarnik, D., A. Zeevi. 2006. Validity of heavy traffic steady-state approximations in generalized Jackson networks. *The Annals of Applied Probability* **16**(1) 56–90.

Gurvich, I. 2013. Validity of heavy-traffic steady-state approximations in multiclass queueing networks: The case of queue-ratio disciplines. *Mathematics of Operations Research* **39**(1) 121–162.

Gurvich, I., J.A Van Mieghem. 2015. Collaboration and multitasking in networks: Architectures, bottlenecks and throughput. *Manufacturing & Service Operations Management* **17**(1) 16–33.

Harrison, J. M., C.V. Mandayam, D. Shah, Y. Yang. 2014. Resource sharing networks: Overview and an open problem. *Stochastic Systems* **4** 1–32. DOI: 10.121413-SSY130.

Hopp, W. J., S. M. R. Iravani, F. Liu. 2009. Managing white-collar work: An operations-oriented survey. *Production and Operations Management* **18**(1) 1–32.

Kang, WN, RJ Williams, et al. 2012. Diffusion approximation for an input-queued switch operating under a maximum weight matching policy. *Stochastic Systems* **2**(2) 277–321.

Krishnamoorthy, A, P.K. Pramod, S.R. Chakravarthy. 2014. Queues with interruptions: A survey. *TOP* **22**(1) 290–320.

Lan, W., T. L. Olsen. 2006. Multiproduct systems with both setup times and costs: Fluid bounds and schedules. *Operations Research* **54**(3) 505–522.

Reiman, M., L. Wein. 1998. Dynamic scheduling of a two-class queue with setups. *Operations Research* **46**(4) 532–547.

Shah, D., D. Wischik. 2012. Switched networks with maximum weight policies: Fluid approximation and multiplicative state space collapse. *Ann. Appl. Prob.* **22**(1) 70–127.

Shah, Devavrat, NS Walton, Yuan Zhong, et al. 2014. Optimal queue-size scaling in switched networks. *The Annals of Applied Probability* **24**(6) 2207–2245.

Simatos, F., N. Bouman, S. Borst. 2014. Lingering issues in distributed scheduling. *Queueing Systems* **77**(2) 243–273.

Stolyar, A. L. 2004. Maxweight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. *Ann. Appl. Prob.* **14**(1) 1–53.

Walton, N. 2015. Concave switching in single-hop and multihop networks. *Queueing Systems* 1–35.