IPSO: A Scaling Model for Data-Intensive Applications

Zhongwei Li, Feng Duan, Minh Nguyen, Hao Che, Yu Lei and Hong Jiang
Department of Computer Science & Engineering
University of Texas at Arlington
Arlington, U.S.
Email: zhongwei.li@mavs.uta.edu, feng.duan@mavs.uta.edu, mqnguyen@mavs.uta.edu,
hche@uta.edu, ylei@uta.edu, hong.jiang@uta.edu

***Abstract*—Today's data center applications are predominantly data-intensive, calling for scaling out the workload to a large number of servers for parallel processing. Unfortunately, the existing scaling laws, notably, Amdahl's and Gustafson's laws are inadequate to characterize the scaling properties of data-intensive workloads. To fill this void, in this paper, we put forward a new scaling model, called In-Proportion and Scale-Out-induced scaling model (IPSO). IPSO generalizes the existing scaling models in two important aspects. First, it accounts for the possible *in-proportion scaling*, i.e., the scaling of the serial portion of the workload in proportion to the scaling of the parallelizable portion of the workload. Second, it takes into account the possible *scale-out-induced scaling*, i.e., the scaling of the collective overhead or workload induced by scaling out. IPSO exposes scaling properties of data-intensive workloads, rendering the existing scaling laws its special cases. In particular, IPSO reveals two new pathological scaling properties. Namely, the speedup may level off even in the case of the fixed-time workload underlying Gustafson's law, and it may peak and then fall as the system scales out. Extensive MapReduce and Spark-based case studies demonstrate that IPSO successfully captures diverse scaling properties of data-intensive applications. As a result, it can serve as a diagnostic tool to gain insights on or even uncover counter-intuitive root causes of observed scaling behaviors, especially pathological ones, for data-intensive applications. Finally, preliminary results also demonstrate the promising prospects of IPSO to facilitate effective resource provisioning to achieve the best speedup-versus-cost tradeoffs for data-intensive applications.**

***Index Terms*—scale-out workload, cloud computing, speedup, performance evaluation, Amdahl's Law, Gustafson's Law**

I. Introduction

Predominant applications in today's datacenters are data-intensive and scale-out by design, based on, e.g., MapReduce [1], Spark [2], and Dryad [3] programming frameworks. For such applications, job execution may involve one or multiple rounds of parallel task processing with massive numbers of tasks and the associated data shards being scaled out to up to tens of thousands of low-cost commodity servers, followed by a serial (intermediate) result merging process. Clearly, from both user's and datacenter provider's perspective, it is imperative to gain good understanding of the scaling properties of such applications so that informed datacenter resource provisioning decisions can be made to achieve the best speedup-versus-cost tradeoffs. Unfortunately, however, the existing scaling laws that have worked well for parallel, high-performance computing, such as Amdahl's law [4], Gustafson's law [5], and Sun-Ni's law [6], are no longer adequate to characterize the scaling properties of data-intensive workloads for two reasons.

First and foremost, the traditional scaling models underlying these laws are exclusively focused on the scaling of the parallelizable portion of the workload or *external scaling* (e.g., the fixed-size, fixed-time, and memory-bounded external scaling models underlying Amdahl's, Gustafson's, and Sun-Ni's laws, respectively), leaving the scaling of the serial portion of the workload or *internal scaling* a constant. Fig. 1 illustrates this, i.e., scaling out to three parallel processing units for the Amdahl's model in Fig. 1(b) and Gustafson's or Sun-Ni's model in Fig. 1(c) from the sequential execution case in Fig. 1(a). While the parallelizable portion of the workload stays unchanged (i.e., fixed-size) and grows by three times (i.e., fixed-time or memory-bounded), respectively, the serial portion of the workload remains unchanged. The rationale behind this assumption is the understanding that the serial portion of a program mostly occurs in the initialization phase of the program, which is independent of the program size [5]. This assumption, however, no longer holds true for data-intensive workloads. This is because as the parallelizable portion of a data-intensive workload increases, so does the serial portion of the workload in general. In other words, the (intermediate) results to be merged in each round of the job execution are likely to grow, in proportion to the external scaling, referred to as *in-proportion scaling* in this paper.

Second, the existing scaling models do not take the possible *scale-out-induced scaling* into account, i.e., the scaling of the collective overhead or workload induced by the external scaling. As being widely recognized (see Section 2 for details), for data-intensive applications, such workloads cannot be neglected in general and they may be induced for various reasons, e.g., task dispatching, data broadcasting, reduction operation, or any types of resource contentions among parallel tasks. Both in-proportion scaling and scale-out-induced scaling are responsible for the scalability challenges facing today's programming frameworks, such as Hadoop and Spark [7].

To overcome the above inadequacies of the existing scaling models, in this paper, we put forward a new scaling model, referred to as In-Proportion and Scale-Out-induced scaling model (IPSO). IPSO augments the traditional scaling models

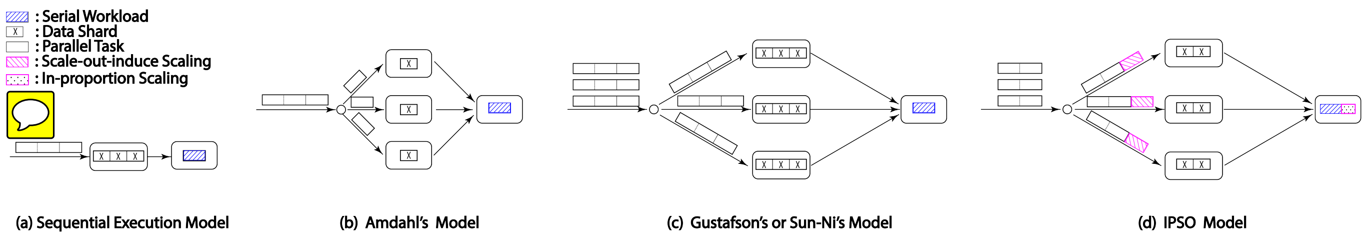


Fig. 1: Speedup models: For data-intensive applications, Sun-Ni's model coincides with Gustafson's model (see Section IV for details).

with the in-proportion scaling and scale-out-induced scaling, as illustrated in Fig. 1(d) (note that the shard size can be one, two, or three), rendering the traditional scaling models and their respective scaling laws its special cases. In particular, IPSO reveals two new pathological scaling properties that are not captured by the existing scaling laws. Namely, the speedup may level off even in the case of the fixed-time workload, and it may peak and then fall as the system scales out, for which Gustafson's law says that the speedup should be unbounded. While the former scaling property is due to the in-proportion scaling, the latter may be attributed to either in-proportion scaling or scale-out-induced scaling. Moreover, the scale-out-induced scaling, in the worst case, may lead to negative speedups, which cannot be captured by the existing scaling laws.

Our extensive case studies for both MapReduce and Spark-based applications demonstrate that while the existing scaling laws fail to capture most of the scaling properties for these applications, IPSO is able to do so for all the cases studied. As a result, IPSO can serve as a diagnostic tool that can gain insights or even uncover counter-intuitive root causes of the observed scaling behaviors, especially, pathological ones, for data-intensive applications. Finally, our preliminary results suggest that as long as the three scaling factors, including the external, internal, and scale-out-induced scaling factors, can be accurately estimated at small problem sizes, the speedups at large problem sizes may be predicted with high accuracy. This sheds light on the possible development of efficient, measurement-based resource provisioning algorithms to achieve the best speedup-versus-cost tradeoffs for data-intensive workloads.

The remainder of the paper is organized as follows. Section II provides the background information to motivate the current work. Section III introduces IPSO. Section IV characterizes the IPSO solution space. Section V presents the application of IPSO to the MapReduce and Spark-based case studies. Finally, Section VI concludes the paper and proposes future research.

II. Background, Related Work and Motivations

The traditional scaling laws for parallel computing were discovered in the context of high performance computing. Amdahl's law [4], Gustafson's law [5] and Sun-Ni's law [6] are the most notable examples of such laws. Recently, extensions of these laws are being proposed, e.g., in the context of multicore processors [8], multithreaded multicore processors [9], power consumption [10], and resource scaling in cloud [11]. However, none of these extensions takes the possible in-proportion scaling or scale-out-induced scaling into account.

Meanwhile, with the advent and proliferation of scale-out, data-intensive applications, rich scaling properties for such applications continue to reveal themselves, most of which, however, cannot be adequately characterized by the existing scaling laws. Here are some examples. It was found [12] that for a fixed-size iterative computing and broadcast scale-out Spark-based workload, the job stops scaling at about $n = 60$, beyond which the speedup decreases due to linear increase of the broadcast overhead, where n is the number of computing nodes for parallel processing. TCP-incast overhead was found to be responsible for the speedup reduction for many big data analytics applications [13]. Centralized job schedulers used in some popular programming frameworks, such as Hadoop and Spark, were found to pose performance bottlenecks for job scaling, due to a quadratic increase of the task scheduling rate as n increases [7]. In fact, a queuing-network-model-based analysis [9] reveals that any resource contention among parallel tasks is guaranteed to induce an effective serial workload, resulting in lower speedup than that predicted by the existing laws.

The scaling analysis of data mining applications [14] reveals that the reduction operations in each merging phase are induced by external scaling, resulting in much lower speedup than that predicted by Amdahl's law. As we shall demonstrate in Section V, even for some simple MapReduce-based applications, including Sort and TeraSort, their scaling properties cannot be captured by the existing scaling laws, largely due to the in-proportion scaling. The Spark-based case studies in Section V further reveal that parallel scaling in both fixed-time and fixed-size dimensions, underlying Gustafson's and Amdahl's laws, respectively, exhibit scaling behaviors that significantly deviate from those predicted by these scaling laws.

The above examples clearly demonstrate the inadequacy of the existing scaling laws in capturing the scaling properties of data-intensive applications. The importance and the urgency of the ability to do so cannot be overemphasized for two main reasons. First, the existing scaling laws may lead to overly optimistic prediction of the scaling performance for data-intensive workloads. They may even make qualitatively incorrect prediction when a pathological situation occurs (see

Section V for examples). In our opinion, the lack of a sound scaling model is largely responsible for the unsettled debate over whether scaling-out is indeed better than scaling-up or not [15]. Second, as the existing scaling laws are increasingly being adopted to not only characterize the scaling properties, but also facilitate resource provisioning for data-intensive workloads [16], it becomes urgent to develop a comprehensive scaling model that can help pinpoint the exact conditions under which the existing scaling laws may be applied.

The importance and the urgency to develop a comprehensive scaling model for data-intensive applications motivate the work presented in this paper.

III. IPSO Modeling

First, we must realize that the main goal of scaling analysis for parallel computing is to capture the scaling properties of the speedup for parallel computing over sequential computing, when the problem size becomes large. A scaling model is considered to be a good one, as long as it captures in a qualitative fashion (e.g., bounded or unbounded, linear or non-linear, monotonic or peaked) major scaling properties of the applications in question. Due to the need to deal with large problem sizes and the tolerability of quantitative imprecision, idealized scaling models that overlook much of the system and workload details are generally adopted, targeting at analytical results that can scale to large problem sizes. The IPSO model is depicted in Fig. 1, together with the Amdahl's, Gustafson's and Sun-Ni's models. Scaling modeling generally involves the modeling of both the system and workload.

System Model: In the same spirit as the existing scaling models, IPSO adopts the same idealized system model that underlies all three existing scaling laws, i.e., Amdahl's, Gustafson's and Sun-Ni's laws. This system model, in the context of data-intensive applications, can be viewed as a homogeneous Split-Merge model with $n + 1$ identical processing units [9], as illustrated in Fig. 1, with $n = 3$. There are n processing units in the split phase, processing the parallelizable portion of the workload in parallel and one processing unit in the merge phase, processing the serial portion of the workload sequentially.

Specifically, the Split-Merge model characterizes the execution of a job composed of one round of parallel task processing with barrier synchronization in the split phase, followed by sequential result merging in the merge phase. Here n is a measure of the degree of scale-out and hence, called *scale-out degree* hereafter. This model can also be applied to the case where there are multiple rounds of the split and merge phases with the same number of processing units in each split phase.

Workload Model: The main effort in developing IPSO is the modeling of the workload. IPSO generalizes and augments the workload models underlying the three speedup laws, hence making them the special cases of IPSO.

For data-intensive applications, the offered workload at each parallel processing unit is proportional to the data shard size at that unit. As a result, as n increases, the total data shard remains to be n (i.e., three as in Fig. 1(b)) and increases by n times (i.e., 9 as in Fig. 1(c)) for the fixed-size and fixed-time/memory-bounded cases, respectively. The IPSO model allows the fixed-size, fixed-time, or anywhere in between as the scale-out degree increases, e.g., doubling the total shard size for the example in Fig. 1(d). In general, the task processing time for the task mapped to processing unit i in the split phase is a random variable, denoted as $T_{p,i}(n)$, serving as a measure of the workload corresponding to task i, for $i = 1, 2, \cdots, n$. As a result, $T_{p,i}(n)$ may grow in (linear) proportion to the size of the data shard mapped to the processing unit i. The processing time for serial result merging in the merge phase is again a random variable, denoted as $T_s(n)$, which, for data-intensive applications, may grow in proportion to the size of the total working data set or total shard size mapped to the split phase, as shown in Fig. 1(d), whereas its counterparts in Fig. 1(b) and (c) stay unchanged. Now, let $W_p(n)$ and $W_s(n)$ represent the total parallelizable and serial portions of the job workload, respectively, and define,

$$W_p(n) = \mathbb{E}[\sum_{i=1}^{n} T_{p,i}(n)] \tag{1}$$

$$W_s(n) = \mathbb{E}[T_s(n)] \tag{2}$$

where $\mathbb{E}[x]$ represents the mean of random variable x. Here $W_p(n)$ and $W_s(n)$ should be interpreted as the average amount of time it takes to process the parallelizable and serial portions of the job workload sequentially using one processing unit[1]. Further define,

$$W_p(n) = W_p(1) \cdot EX(n) \tag{3}$$

$$W_s(n) = W_s(1) \cdot IN(n) \tag{4}$$

where $EX(n)$ and $IN(n)$ are called *external* and *internal* scaling factors, corresponding to the scaling of the parallelizable and serial portions of the workload, respectively. These scaling factors enable in-proportion scaling. We further define in-proportion scaling ratio, $\epsilon(n)$, as follows,

$$\epsilon(n) = \frac{EX(n)}{IN(n)} \tag{5}$$

As we shall see shortly, a rich set of scaling properties can be uncovered by properly selecting this ratio.

Now we further introduce the scale-out-induced workload shown in Fig. 1(d) and denote it as $W_o(n)$. $W_o(n)$ represents the collective overhead induced by the scale-out itself, e.g., due to job scheduling, data shard distribution, and the queuing effect for result merging. We define,

$$W_o(n) = \frac{W_p(n)}{n} q(n) \tag{6}$$

[1]Note that by definition, the sequential job execution does not generate scale-out-induced workload, hence $W_o(n)$ does not appear in the numerator.

where $q(n)$ is called *scale-out-induced scaling factor*, which is a non-decreasing function of n and equals zero at $n = 1$. It captures the effective workload induced solely by the scale-out degree n, independent of the task workload size. In contrast, its coefficient, $\frac{W_p(n)}{n}$, i.e., the per-task workload, captures the possible dependency of $W_o(n)$ on the task workload size. For example, the data shard distribution overhead grows with both n and the task workload size or data shard size.

Finally, with the barrier synchronization and randomness of parallel task processing times, the mean job response time with respect to parallel task processing is given by the slowest task, i.e., $\mathbb{E}[\max\{T_{p,i}(n)\}]$.

With the above scaling model, the speedup, $S(n)$, can then be expressed as follows,

$$S(n) = \frac{W_p(n) + W_s(n)}{\mathbb{E}[\max\{T_{p,i}(n)\}] + W_s(n) + W_o(n)} \tag{7}$$

While the numerator is the average amount of time it takes to process the entire job workload sequentially using one processing unit, the denominator is the average amount of time it takes to process the entire job workload in parallel with n processing units, plus the workload due to scale-out-induced scaling. Substituting Eqs. (1)-(6) into Eq. (7), we have,

$$S(n) = \frac{\eta EX(n) + (1-\eta)IN(n)}{\frac{\mathbb{E}[\max\{T_{p,i}(n)\}]}{\mathbb{E}[T_{p,1}(1)]+\mathbb{E}[T_s(1)]} + (1-\eta)IN(n) + \frac{\eta EX(n)q(n)}{n}} \tag{8}$$

where η is the percentage of the parallelizable portion of the job workload at $n = 1$, i.e.,

$$\eta = \frac{W_p(1)}{W_p(1) + W_s(1)} \equiv \frac{\mathbb{E}[T_{p,1}(1)]}{\mathbb{E}[T_{p,1}(1)] + \mathbb{E}[T_s(1)]} \tag{9}$$

An executable, sequential job execution model must be defined to allow the numerator in Eq. (7) or (8) to be measurable in practice. It will be given in Section IV, after the workload types are defined (i.e., Eq. (13)).

IV. IPSO Solution Space Characterization

The IPSO workload model developed above is a statistic model that accounts for the possible randomness of the task execution times. The statistic modeling is important in practice if the scaling analysis attempts to capture the scaling properties of an application both qualitatively and quantitatively. For example, to capture the impact of long-tail effects of task service time on the speedup performance, e.g., due to stragglers [17] or the possible task queuing effects [18], the mean job response time for the split phase must be characterized statistically by $\mathbb{E}[\max\{T_{p,i}(n)\}]$ (see Eq. (8)). However, since $\mathbb{E}[\max\{T_{p,i}(n)\}]$ is upper bounded as the problem size in terms of n becomes large, given that the tail length of the task response time must be finite in practice, whether to use statistic or deterministic modeling will not make a difference in terms of capturing the qualitative scaling properties of an application. The reason that we formulate IPSO as a statistic model is to allow accurate scaling prediction that may serve as the basis for future development of a measurement-based job resource provisioning approach for data-intensive applications. So in the rest of the paper, we shall focus on the deterministic model only for simplicity and ease of presentation. The deterministic IPSO refers to a special case where $T_{p,i}(n) = t_p(n)\ \forall\ i$ and $T_s(n) = t_s(n)$. Here $t_p(n)$ and $t_s(n)$ are deterministic functions of n. In this case, $\mathbb{E}[\max\{T_{p,i}(n)\}] = t_p(n)$. Hence, from Eq. (8), we have,

$$S(n) = \frac{\eta EX(n) + (1-\eta)IN(n)}{\frac{\eta EX(n)}{n}(1 + q(n)) + (1-\eta)IN(n)} \tag{10}$$

where η in Eq. (9) can be rewritten as,

$$\eta = \frac{t_p(1)}{t_p(1) + t_s(1)} \tag{11}$$

Clearly, by viewing $W_p(n)$, $W_s(n)$ and $W_o(n)$ as the sum of the corresponding workloads in all rounds, the above IPSO model can be applied to the case involving multiple rounds of the same scale-out degree, n.

Relation to the well-known speedup laws: With the notations defined in this paper, the three well-known speedup laws can be written as,

$$S(n) = \begin{cases} \frac{1}{\frac{\eta}{n} + (1-\eta)}\ , & \text{Amdahl's law;} \\ \eta n + (1-\eta)\ , & \text{Gustafson's law;} \\ \frac{\eta \bar{g}(n) + (1-\eta)}{\frac{\eta \bar{g}(n)}{n} + (1-\eta)}\ , & \text{Sun-Ni's law.} \end{cases} \tag{12}$$

The scaling properties for these laws can be derived from Eq. (10), by letting $IN(n) = 1$ and $q(n) = 0, \forall\ n$, i.e., without considering the possible in-proportion scaling and scale-out-induced scaling, and,

$$EX(n) = \begin{cases} 1\ , & \text{fixed-size: Amdahl's law;} \\ n\ , & \text{fixed-time: Gustafson's law;} \\ \bar{g}(n)\ , & \text{memory-bounded: Sun-Ni's law.} \end{cases} \tag{13}$$

meaning that the total parallelizable portion of the workload stays unchanged for fixed-size workload; linearly increases for fixed-time workload; and scales with the memory size for memory-bounded workload, respectively, as the system scales out or n increases. Here $\bar{g}(n)$ is the external scaling factor, constrained by the total memory space, which in turn, is determined by n, assuming that the maximum affordable memory space to accommodate part of the working data set at each parallel processing unit is a given, e.g., 128 MB [6]. For all the cases studied in this paper where the working data sets are memory bounded, $\bar{g}(n) \approx n$ with high precision (see Fig. 6), i.e., almost the same as that for the fixed-time workload. For this reason, we assume that the Gustafson's and Sun-Ni's models are the same (see Fig. 1(c)) in the context of data-intensive applications, and in what follows, we exclusively focus on fixed-size and fixed-time workload types only.

A Remark: We observe that in the context of data-intensive workloads, the fixed-size and fixed-time workload models capture two extreme scenarios, i.e., resource-abundant and resource-constrained, respectively. By resource-abundant, we mean that the parallelizable portion of the workload can be processed in its entirety by one processing unit. In this case, one is interested in characterizing the scaling behaviors when the fraction of the parallelizable workload on each processing unit decreases as the scale-out degree, n, increases, i.e., the Amdahl's case. By resource-constrained, we mean that each processing unit can only handle a fraction of the total parallelizable portion of the workload, e.g., the case when the memory allocated to each processing unit is fully occupied by the data shard assigned to it. In this scenario, the workload linearly grows with n, i.e., the Gustafson's case. In general, however, as the system scales out, a data-intensive workload may scale in either way or anywhere in between. As a result, a comprehensive scaling model should be able to cover both fixed-time and fixed-size workload types, as is the case for IPSO.

Finally, with the workload types defined in Eq. (13), now we are in a position to define an executable sequential job execution model underlying the numerator in Eq. (7) or (8). For fixed-time workload, our sequential job execution model works as follows. It first runs n tasks in the split phase sequentially using one processing unit. It then merges task results in the merging phase using another processing unit. Since the merging phase may not start until all the tasks finish, due to barrier synchronization, this model is equivalent to using only one processing unit to execute the job sequentially, which agrees with the common understanding of what a sequential execution model is supposed to be. For fixed-size workload, the same sequential job execution model applies. The only difference is that now $n = 1$, i.e., only one task is executed in the map phase over the entire working data set that is assumed to fit into the memory in a single processing unit. This model is in line with the sequential job execution model, implicitly used by Amdahl to evaluate the numerator in the speedup formula, i.e., the entire workload is executed as one task using one processing unit.

Analysis of scaling properties of IPSO: We are interested in exploring major scaling properties of IPSO in the entire solution space spanned in three dimensions, including $EX(n)$, $IN(n)$, and $q(n)$. In other words, the scaling behaviors of IPSO are fully captured so long as these factors are known. As explained before, for scaling analysis, one is interested in the qualitative scaling behaviors of the speedup when n becomes large. In this case, $\epsilon(n)$ in Eq. (5) can be written approximately as (i.e., only the highest order term is kept),

$$\epsilon(n) \approx \alpha n^{\delta} \qquad \text{as } n \text{ becomes large.} \tag{14}$$

where α is a nonnegative coefficient and δ determines the relative order of "speed" of external scaling versus internal scaling. Likewise, $q(n)$ can be approximated as follows,

$$q(n) \approx \beta n^{\gamma} \qquad \text{as } n \text{ becomes large.} \tag{15}$$

where β is a nonnegative coefficient and $\gamma \geq 0$. Here $\gamma = 0$ corresponds to the case without scale-out-induced workload, i.e., $q(n) = 0$.

With Eqs. (14), (15) and (5), Eq. (10) can be rewritten as follows,

$$S(n) \approx \frac{\eta \alpha n^{\delta} + (1-\eta)}{\eta \alpha n^{\delta-1}(1+\beta n^{\gamma}) + (1-\eta)} \tag{16}$$

Note that for the workload without a serial portion, i.e., $W_s(n) = 0$ or $\eta = 1$, Eqs. (14) and (5) are undefined. In this case, from Eq. (10), we have,

$$S(n) = \frac{n}{1+\beta n^{\gamma}} \tag{17}$$

With these two formulas, we are now ready to explore the entire IPSO solution space. We consider fixed-time and fixed-size workload types, separately.

Fixed-time workload type ($EX(n) = n$): In this case, $0 \leq \delta \leq 1$. This is because in practice, as the parallel portion of the workload scales up linearly fast, $IN(n)$ is unlikely to scale down or scale up superlinearly fast. From Eqs. (16) and (17), we identify the following four distinct types of speedup scaling behaviors, as depicted in Fig. 2:

- I_t: This type is Gustafson-like, i.e., the speedup linearly grows and degenerates to Gustafson's law at $\alpha = 1$. As shown in Fig. 2, it occurs when there is no scale-out-induced workload (i.e., $\gamma = 0$, or equivalently, $q(n) = 0$) and either $\delta = 1$ (i.e., with no internal scaling) or in the absence of the serial workload (i.e., $\eta = 1$);
- II_t: Speedup grows sublinearly but still unbounded. It occurs when $q(n)$ grows slower than linear, i.e., $\gamma < 1$, and either $0 < \delta \leq 1$ or $\eta = 1$;
- III_t : This type is pathological, i.e., the speedup grows monotonically but is upper-bounded. There are two sub-types here, i.e., $III_{t,1}$ and $III_{t,2}$, with distinct upper bounds, as depicted in Fig. 2, corresponding to sublinear and linear scale-out scalings, respectively;
- IV_t: This type is even more pathological as the speedup peaks and falls, and finally enters negative speedup region. It occurs when $q(n)$ scales up superlinearly fast, i.e., $\gamma > 1$, regardless of how the other scaling factors behave.

Fixed-size workload type ($EX(n) = 1$): In this case, $\delta = 0$. This is because without scaling the parallel portion of the workload, the serial portion of the workload will not scale, i.e., $IN(n) = 1$, and any workload added as n increases should be viewed as part of $W_o(n)$, i.e., scale-out-induced workload. Again, from Eqs. (16) and (17), four distinct types of speedup scaling behaviors are identified, as depicted in Fig. 3 (note that although they look the same as their counterparts in Fig. 2, the associated scaling factors are different):

- I_s: $S(n) = n$. it occurs when there is no scale-out-induced workload (i.e., $\gamma = 0$) and $\eta = 1$ (i.e. no serial portion of the workload), a very special case;

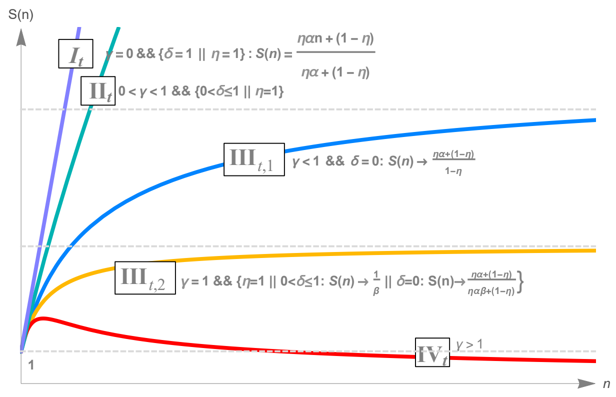


Fig. 2: Four distinct IPSO scaling behaviors for the fixed-time workload type: I_t: Gustafson-like linear scaling; II_t: unbounded sublinear scaling; III_t: pathological, upper-bounded scaling; and IV_t: pathological, peaked scaling.

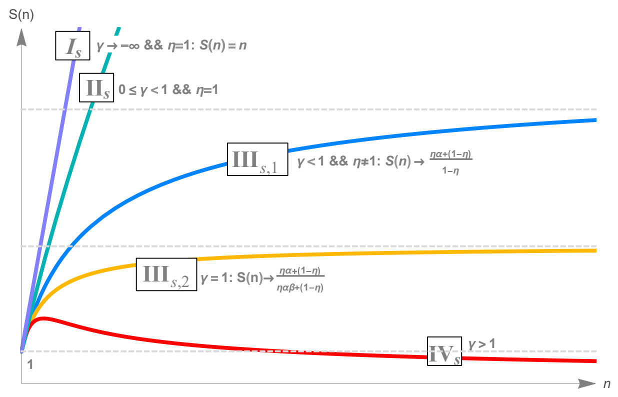


Fig. 3: Four distinct IPSO scaling behaviors for the fixed-size workload type: I_s: linear scaling; II_s: unbounded sublinear scaling; III_s: Amdahl-like upper-bounded scaling; and IV_s: pathological, peaked scaling.

- II_s: Speedup grows sublinearly and unbounded. It occurs when $q(n)$ grows slower than linear, i.e., $\gamma < 1$ and $\eta = 1$, also a special case;
- III_s : It is Amdahl-like. It grows monotonically and is upper-bounded. Again, it is composed of two subtypes, i.e., $III_{s,1}$ and $III_{s,2}$, with distinct upper bounds, similar to $III_{t,1}$ and $III_{t,2}$, as depicted in Fig. 3. Clearly, Amdahl's law is a special case of $III_{s,1}$ at $\gamma = 0$ and $\alpha = 1$;
- IV_s: This type is pathological and behaves similar to IV_t given in Fig. 2. It occurs when $q(n)$ scales up superlinearly fast, regardless of how the other scaling factors behave.

In summary, both fixed-time and fixed-size workloads may suffer from pathological types of scaling, i.e., IV_t and IV_s, respectively, which by all means, should be avoided. The root cause for both IV_t and IV_s points to the superlinear scaling of $q(n)$, often seen in the case related to centralized job scheduling and data shard broadcasting. A case study of such kind will be given in the following section. The next scaling behavior that is also pathological and should be avoided is III_t. This is because I_t (or Gustafson's law) and II_t suggest that unbounded speedup should be achievable for the fixed-time workload type. On the other hand, upper-bounded speedup or III_s has long been understood to be inevitable for the fixed-size workload type, since the discovery of Amdahl's law. This is because unbounded speedup, i.e., I_s and II_s, for this workload type only occur under very special circumstances. Nevertheless, the achievable upper bound for III_s may vary and effort should be made to attain the highest possible bound.

V. Application of IPSO to Scaling Analysis of Data-Intensive Applications

The main focus of this section is to test the ability of IPSO in capturing the scaling properties, and hence, its suitability in serving as a diagnostic tool for scaling analysis of data-intensive applications. As a byproduct, the ability of IPSO to predict the scaling properties for some simple cases is also explored, demonstrating the promising potentials of IPSO to facilitate effective resource allocation for data-intensive applications. This study includes a total of nine cases, including four Map-Reduce-based and four Spark-based case studies, performed on the Amazon EC2 cloud, and one Spark-based case study, extracted from [12]. This allows a sufficiently wide range of both single-stage (i.e., single-round) and multi-stage applications with rich scaling properties to be uncovered. These case studies also reveal the inability of the existing scaling laws in characterizing the scaling properties of data-intensive applications in general. In fact, out of the nine cases studied, only two simplest MapReduce cases follow the existing scaling laws. Finally, we note that the data presented are average results of multiple experimental runs.

A. *Single-Stage Cases*

This section focuses on the cases with only one round of job execution. They are either fixed-time or fixed-size, which are studied separately.

Fixed-Time Workload: The cases studied include three representative micro benchmarks from the HiBench suite [19], a widely adopted benchmark suite for Hadoop, i.e., WordCount, Sort, and TeraSort [20]. The working data sets for WordCount and Sort are randomly generated text, drawn from a UNIX dictionary that contains 1000 words. TeraSort uses a working data set derived directly from an embedded TeraGen program. Also included in this study is a QMC Pi program from Apache Hadoop examples [21]. This program uses Quasi Monte Carlo method (QMC) to estimate the value of π. These four applications are chosen because they are among the most widely studied MapReduce-based applications that involve a single round of parallel processing and merging.

On the hardware side, all of the four experiments were carried out on Amazon EC2 cloud with EMR (elastic MapReduce) support. An m4.4xlarge virtual machine (VM) instance is chosen as the master node and m4.large VM instances as processing units. We configure the resource manager to launch only one container per processing unit.

And all the MapReduce jobs in these experiments are configured as involving a single reducer with synchronization barrier.

Results: The measured speedups for the four cases are presented in Fig. 4, together with the ones predicted by Gustafson's law. We make the following observations.

First, the QMC case in Fig. 4(a) matches Gustafson's law well. For this case, there is no serial workload, i.e., $\eta = 1$ and the fact that it matches Gustafson's law well means that it must belong to I_t, as depicted in Fig. 2. This further implies that there is little scale-out-induced workload involved, since $\gamma = 0$ for I_t.

Second, the WordCount case in Fig. 4 (b) must be either I_t or II_t, as it is close to linear growth as predicted by Gustafson's law, but more data samples at larger scale-out degree are needed to be certain which one it belongs to. Nevertheless, at least, we know from I_t and II_t that for this case, the scale-out-induced workload is very small and the speedup is very likely to be unbounded, i.e., a benign case in terms of scaling.

Third, it is clear that the Sort and TeraSort cases in Fig. 4 (c) and (d), respectively, significantly deviate from that predicted by Gustafson's law and fall into $III_{t,1}$ or $III_{t,2}$, or somewhere in between. Our detailed scaling factor analysis, as will be given shortly, indicates that both are more on the $III_{t,1}$ side than the $III_{t,2}$ side, since δ is close to zero and γ is likely to be small.

The above case studies clearly demonstrate that IPSO can provide significant insights on the possible root causes of the scaling behaviors for data-intensive applications. However for some cases, to exactly pin down the root cause, the exact scaling parameters, e.g., δ and γ, must be estimated. In what follows, we make a first attempt to estimate these parameters for the above cases. The aim is twofold: (a) demonstrating the promising potentials of IPSO to serve as a capable means for scaling prediction and hence, resource allocation; and (b) exposing the challenges in this pursuit.

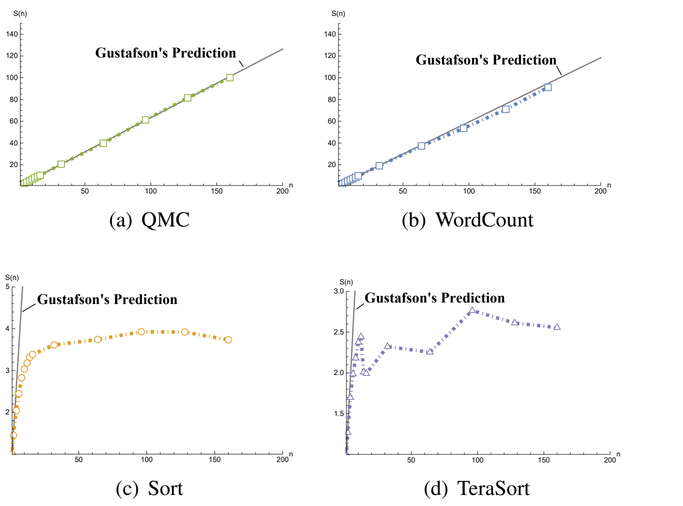


Fig. 4: Measured speedups for the four selected intel-HiBench micro benchmarks, together with the speedups predicted by Gustafson's law.

Scaling Prediction: Regardless of the implementation details, every MapReduce job execution time can be roughly broken down into four parts: (a) the execution environment initialization and job scheduling; (b) the map phase (i.e., the split phase); (c) the communication between map and reduce phases; and (d) the reduce phase (i.e., the merge phase). The reduce phase can be further divided into a shuffle stage (the stage during which the reducer pulls all mappers' results from DFS), a merge stage (the stage when the reducer merges specific amount of mapper results to get intermediate results), and a reduce stage (the last merging process that produces the final results). With reference to this background information, the following describes how we identify, based on measurement, the scaling behaviors of the three workloads, $W_o(n)$, $W_p(n)$, and $W_s(n)$, in terms of three scaling factors, $q(n)$, $EX(n)$, and $IN(n)$, respectively.

First, by comparing the execution times of individual non-workload processing parts and stages in the scale-out execution against the corresponding times in the sequential execution, the scale-out-induced workload, $W_o(n)$, if any, can be identified (i.e., the overheads that grow with n in the scale-out execution, which are absent from the sequential job execution). $W_o(n)$ is more likely to come from parts (a) and (c). By detailed measurement, we find that $W_o(n)$, if any, is negligibly small for all four cases, due to the dominance of parts (b) and (d) and also the fact that the workload is the fixed-time one. We then further inspect the shuffle stage to see if there is any discrepancies between the two, which turns out to be negligible as well.

The next two steps are to measure $EX(n)$ and $IN(n)$. To this end, we first note that part (b), i.e., the map phase, is the sole contributor to the parallel processing phase and the rest can be attributed to the sequential merging phase. With this understanding, we estimate $EX(n)$ and $IN(n)$ by curve fitting based on the measured data at problem sizes no greater than $n = 16$ for WordCount, Sort, and QMC Pi.

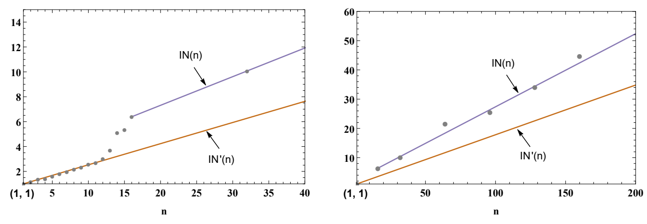
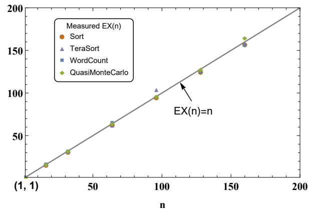


Fig. 5: The internal-scaling factor of TeraSort demonstrates a step-wise property. As shown there are two distinct functions: a linearly increasing function $IN'(n)$ with a slower growth pace for small problem size, and $IN(n)$ at a faster growing speed when the problem size grows bigger.

For TeraSort, we use the measured data between $n = 16$ and $n = 64$. This is because for TeraSort, the data input size which grows linearly with n exceeds the preconfigured reducer memory size (∼2GB) at about $n = 15$, when the disk I/O is involved and the internal scaling factor is burst by over 30%

with its slope increasing from 0.15 to 0.25 (Fig. 5). This phenomenon is reflected in Fig. 4(d) where there is a small surge of the speedup around $n = 15$ and then falls back before it grows again. It indicates that for scaling factor prediction, one must monitor the possible program execution environment changes as the problem size increases, e.g., memory capacity overflow, surging queuing delay, or onset of network or I/O bottleneck.

The predicted $IN(n)$ and $EX(n)$, together with the measured ones for the problem size reaching $n = 160$ are depicted in Fig. 6. As shown in the figure, the parallelizable workloads are memory bounded (i.e., using maximal block size of 128 MB per processing unit) and $EX(n)$ closely follows fixed-time workload, i.e., $EX(n) \approx n$, as expected. This confirms our earlier claim that a memory-bounded workload is no different from a fix-time workload.



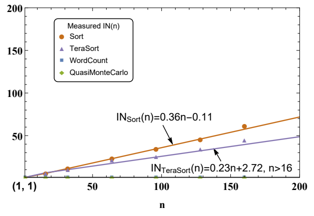
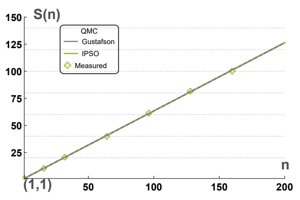


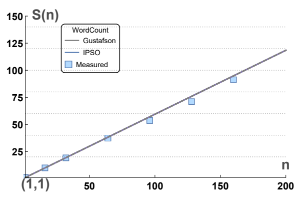
Fig. 6: $EX(n)$ and $IN(n)$ for the four cases. For all the cases, $EX(n) \approx n$, agreeing with the Gustafson's assumption on the external scaling (see Eq. (13)).

For $IN(n)$ in Fig. 6, while WordCount and QMC pi follow the existing workload model, i.e., $IN(n) = 1$, both Sort and Terasort do not. For both applications, the predicted $IN(n)$'s based on linear regression match the measured data well.

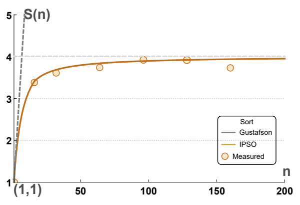
By using both predicted and measured $EX(n)$ and $IN(n)$ above, as well as measured $\mathbb{E}[max\{T_{p,i}(n)\}]$, $\mathbb{E}[T_{p,1}(1)]$ and $\mathbb{E}[T_s(1)]$ as input into Eqs. (9) and (8), one arrives at the speedups for all four cases, as depicted in Fig. 7. As one can see, overall, IPSO predicts the scaling properties for all four cases very well.



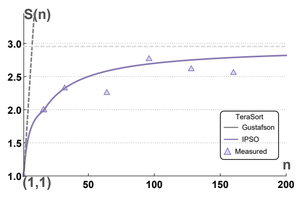
(a) QMC



(b) WordCount



(c) Sort



(d) TeraSort

Fig. 7: The speedups based on IPSO, measurement, and Gustafson's law for the four cases in the case of the fixed-time workloads

With all the scaling factors captured, IPSO can now help characterize the scaling properties for the above four cases with much higher precision than before. First, it is now clear that the in-proportion scaling has led to the $III_{t,1}$ pathological scaling or upper bounded speedup for Sort and Terasort, which cannot be predicted by the existing scaling laws. For example, for TeraSort, even though the scaling ratio is high, i.e., $\epsilon(n) = 4.3$, meaning that the external scaling is more than 4 times faster than the internal scaling, the speedup for TeraSort is upper bounded by the value of 3. In other words, even with a relatively small internal scaling, compared with the external scaling, the speedup becomes quite limited and upper bounded.

The above examples demonstrate the promising prospects of IPSO for scaling prediction and hence, resource provisioning. In the meantime, we also realize that to this end, it entails detailed understanding and analysis of the applications and execution procedures. Clearly, it is very unlikely that an one-size-fits-all prediction solution exists. Instead, the IPSO prediction solution may need to be developed on a per programming framework and per type of applications basis.

Fixed-size Workload: Unfortunately, for all the four cases studied so far, in the case of fixed-size workload, as the scale-out factor n grows beyond 8, the parallel task response times in the map phase drop to subseconds, which cannot be measured, since in our experiments the precision of measurement is one second. Hence, instead of studying the four cases, we consider the following one.

Collaborative Filtering: As explained in [12], some iterative MapReduce machine learning applications in Spark may include significant data broadcasts from the master node to all the worker nodes per iteration. In particular, [12] describes and provides the experimental data for a collaborative filtering application (see [12] for detailed description of this application). In each iteration, there are two feature vectors to be updated alternately, involving two rounds of broadcast and two Map phases with barrier synchronization. For this application, each broadcast is induced as a result of the system scale-out, generating scale-out-induced workload. Moreover, the lack of a reduce phase in each iteration means that there is no sequential merging phase or $W_s(n) = 0$. Since the scale-out degrees for both rounds are the same, IPSO can be applied as aforementioned. Namely, by combining the corresponding workloads in the two rounds into one, each iteration agrees with our IPSO model, and hence, experimental data for each iteration in the form of histograms, as given in Fig. 3 in [12], can be leveraged by IPSO for the scaling analysis of this application. We are able to convert the experimental data in [12] into a table format in Table I and then the data are plotted in Fig. 8, together with the matched curves based on nonlinear regression.

First, with $W_o(n)$ given in Fig. 8(a) and according to Eqs. (6) and (15) (note that $W_p(n) = W_p(1) = \mathbb{E}[T_{p,1}(1)]$ is a constant for the fixed-size workload), we have, $\gamma = 2$. By inspecting Fig. 3, it becomes clear that the speedup for

TABLE I: Measured external and scale-out-induced workloads for Collaborative Filtering

n	$\mathbb{E}[\max\{T_{p,i}(n)\}]$	$W_o(n)$
10	209.0	5.5
30	79.3	17.7
60	43.7	36.0
90	31.1	54.3

this case must fall into IV_s, the worst-case scenario, rather than the best-case scenario, i.e., I_s, as would be predicted by Amdahl's law (note that $\eta = 1$ as there is no serial workload).

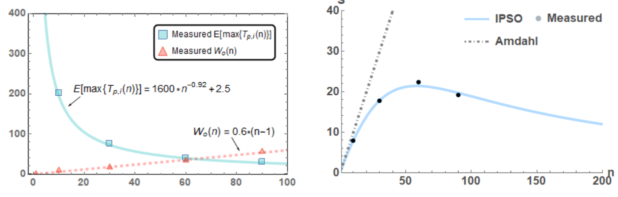
To verify that IPSO indeed predicts the scaling property for the current case accurately, in what follows, we calculate $S(n)$ based on the statistic version of the IPSO model.

First, given $W_p(n) = W_p(1) = \mathbb{E}[T_{p,1}(1)]$, the speedup, $S(n)$, for this application, according to Eq. (7), can then be written as follows,

$$S(n) = \frac{\mathbb{E}[T_{p,i}(1)]}{\mathbb{E}[\max\{T_{p,i}(n)\}] + W_o(n)} \tag{18}$$

By extrapolating the matched curve for $\mathbb{E}[\max\{T_{p,i}(n)\}]$ in Fig. 8(a) to $n = 1$, we have $\mathbb{E}[T_{p,1}(1)] = 1602.5$. Finally, $S(n)$ is evaluated by substituting the matched functions in Fig. 8(a) into Eq. (18).

Fig. 8(b) depicts the measured speedup and IPSO speedup, along with the speedup predicted by Amdahl's law. As one can see, both measured and IPSO speedups confirm that the scaling behaviors follow IV_s, rather than I_s, as predicted by Amdahl's law.



(a) (b)

Fig. 8: The measured and IPSO speedups, together with that predicted by Amdahl's law for Collaborative Filtering.

The scaling properties, depicted in Fig. 8(a), is pathological and disappointing. It simply states that even in the absence of the serial portion of the workload, the linear speedup scaling, predicted by Amdahl's law, is not guaranteed. The dismal speedup, 22, at its peak and continued trending towards zero, as predicted by IPSO, indicates that the scale-out-induced scaling can pose serious threats to the scalability of scale-out applications. Collaborative Filtering deals with a fixed problem size, i.e., $EX(n) = 1$, meaning that scaling out beyond $n = 60$ can only do harm to the parallel computing, hence setting a hard scale-out degree upper bound, beyond which the parallel computing performance deteriorates.

B. Multi-Stage Cases

In this section, IPSO is applied to the analysis of Spark-based applications, involving multiple stages/rounds of parallel task processing per job execution, where the scale-out degree may vary from one stage to another. However, as we mentioned earlier, the IPSO model can only be directly applied to the multi-stage cases with the same scale-out degree.

Fortunately, in the Spark-based program, two configuration parameters, i.e., the problem size, N, and parallel degree, m, must be set. Here N is the nominal number of tasks to be executed in a stage and m the nominal number of executors (i.e., processing units) to run in parallel in a stage. In general, the actual numbers of tasks and executors in the first stage are indeed equal to N and m, respectively. In other words, each of the m executors in the first stage needs to execute N/m tasks sequentially. Although the numbers of tasks and executors in the subsequent stages may not be equal to their respective nominal counterparts, they are strong functions of these counterparts. This implies that the three workloads for Spark-based applications can be defined in terms of N and m, i.e., $W_p = W_p(N,m)$, $W_s = W_s(N,m)$, and $W_o = W_o(N,m)$ in general.

To allow IPSO to be applied to the cases with multi-stage without modification, we simply define the fixed-time and fixed-size cases as when N/m and N are fixed, respectively, while scaling m, i.e., the scale-out degree, $n = m$. By doing so, all the above three workloads become functions of n only for either case, the same as the ones defined in IPSO. Hence, all the results derived from IPSO apply to the Spark-based multi-stage cases. In what follows, we perform case studies for four representative Spark benchmarks.

We deployed the Intel HiBench suite on EC2 with the EMR (Elastic MapReduce) service and then launched four Spark benchmarks including three machine learning applications:*Bayes Classifier (Bayes)*, *Random Forest (RF)*, *Support Vector Machine (SVM)*, and one graph application (*NWeight*). The hardware configuration is cloned from the previously stated MapReduce experiment. All the instances are preconfigured with sufficient storage capacity (100GB per-node) and bandwidth ($\geq$ 450 Mbps). Executor reuse and logging are enabled for performance metrics collection. All the input data sets are generated by the respective data generators provided by this benchmark suite. We then extract the execution latencies for all stages from the application's Log file to derive the speedup. This is done by tracing the timestamps for each stage in the Spark Log files, which are available in the *JSON* format.

To identify the scaling properties using IPSO, we are interested in the speedups projected onto two different dimensions, i.e., the fixed-time (with N/m fixed) and fixed-size (with N fixed) dimensions, while scaling $n = m$ as depicted in Fig. 9 and Fig. 10 respectively. Along with the data points, we also plotted the projected curves of the matched two-dimensional surfaces as functions of N and m based on nonlinear regression for the ease of identification of trending.

First, one notes that for fixed-time workloads, as depicted in Fig. 9, the larger the per executor load level, $\frac{N}{m}$, the higher the speedup is. In other words, the speedup curve at $\frac{N}{m} = 4$ is higher than that at 2, which in turn, is higher than that at 1. The detailed analysis of the code paths indicates that this

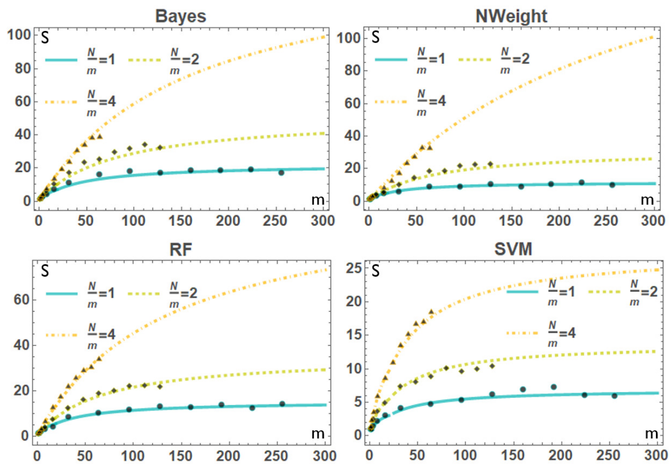


Fig. 9: Fixed-time dimension

is due to the increased percentage of the scale-out-induced workload as the number of tasks per executor decreases. More specifically, the scheduling and deserialization time (i.e., the communication cost) of the first wave of tasks outweigh the following waves. Hence, further increasing per-node workload level effectively reduces the scale-out induced workload per task. This however, by no means suggests that reducing the parallel degree always improves speedup performance. In fact, our study shows that the speedup at $\frac{N}{m} = 8$ is lower than that at $\frac{N}{m} = 4$. This is because the per-node workload level is constrained by the available resource at the node level. For example, insufficient RAM may cause the persistent RDDs to be spilled to the local disk, or even trigger increased task failure rate, leading to the rollback to the previous stage and hence poor performance. In other words, the optimal scale-out level, or parallel degree m is determined by both the workload size and the resource availability at individual executors.

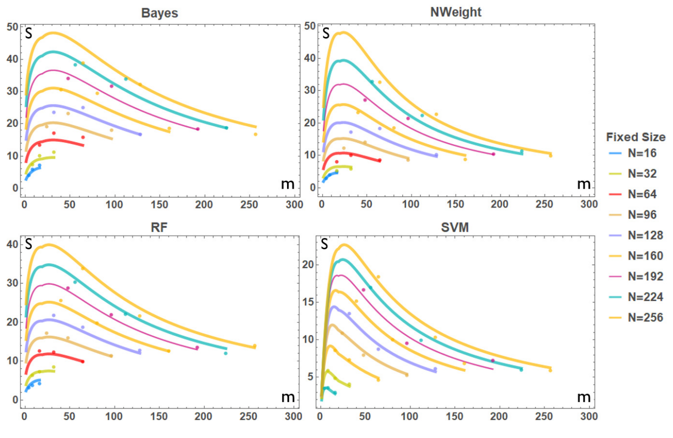


Fig. 10: Fixed-size dimension

The scale-out-induced workload is also responsible for the degradation of the speedups from the ideal Gustafson-like I_t to at best II_t for large $\frac{N}{m}$ and III_t for smaller $\frac{N}{m}$. All four cases share similar scaling properties.

Second, for the fixed-size workload dimension, as depicted in Fig. 10, as the problem size N becomes large, the speedups for all four cases peak and then fall as n increases, agreeing with the scaling behavior of IV_s, the pathological one. This is in stark contrast with that predicted by Amdahl's law, or III_s. This is due to the strong scale-out induced overhead, as discussed above. Again, all four cases share similar scaling behaviors.

In summary, the above case studies clearly demonstrate that IPSO can serve as a diagnostic tool for the scaling analysis of data-intensive applications. Specifically, the following diagnostic procedure is recommended:

1) Determine the use case scenario, i.e., the fixed-time or fixed-size workload;
2) For given workload type, measure the speedup as the scale-out degree increases;
3) Plot the speedup data points versus scale-out degree, maybe together with a matched curve from nonlinear regression as a guide;
4) Compare the trending of the measured speedup with either Fig. 2 or Fig. 3, depending on the workload type to identify closely matched scaling types;
5) If the matched scaling type is I_t (I_s), II_t (II_s), or IV_t (IV_s), the root causes of the scaling behaviors are readily identified and understood based on the analysis in Section IV. If, however, the matched type is III_t (III_s), go to the next step to further identify which sub-type, $III_{t,1}$ ($III_{s,1}$) or $III_{t,2}$ ($III_{s,2}$) it belongs to;
6) Carry out a detailed analysis and measurement of the scaling parameters, δ and γ, to pin down the exact sub-type it belongs to, which however, may need to be done case by case.

Finally, the source code and executables for all the case studies are publicly available at our code repository [22].

VI. Conclusions and Future Work

In this paper, we proposed a new scaling model for scale-out, data-intensive applications, called In-Proportion and Scale-Out-induced scaling model (IPSO). IPSO sets itself apart from the existing scaling models in two important aspects. First, it takes the scaling of the serial portion of a workload into account, referred to as *in-proportion scaling*. Second, it accounts for the possible overheads induced by the scale-out, resulting in what we call *scale-out-induced scaling*. Both in-proportion and scale-out-induced scalings were observed in scale-out applications, but had not been formally defined until this work. MapReduce-and-Spark-based application case studies demonstrate that IPSO can serve as a powerful diagnostic tool for the scaling analysis of data-intensive applications.

As our future work, we plan to develop measurement-based resource provisioning algorithms for data-intensive workloads based on the prediction ideas behind IPSO. The key is to find a solution as to how to quickly estimate the two scaling parameters, δ and γ. The research in this direction is underway.

VII. Acknowledgments

This work is supported by the US NSF under Grant No. XPS-1629625 and SHF-1704504.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in Proceedings of the 6th Symposium on Operating Systems Design and Implementation - OSDI '04, 2004, pp. 137–150. [Online]. Available: http://www.usenix.org/events/osdi04/tech/dean.html

[2] M. Zaharia, "An Architecture for Fast and General Data Processing on Large Clusters," Ph.D. dissertation, University of California, Berkeley, 2013. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-11.pdf

[3] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed Data-parallel Programs from Sequential Building Blocks," in Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, ser. EuroSys '07, 2007, pp. 59–72.

[4] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in Proceedings of Am. Federation of Infomation Processing Societies Conf. ACM, 1967, pp. 483–485.

[5] J. L. Gustafson, "Reevaluating Amdahl's law," Communications of the ACM, vol. 31, no. 5, pp. 532–533, 1988.

[6] X. H. Sun and Y. Chen, "Reevaluating Amdahl's Law in the multicore era," Journal of Parallel and Distributed Computing, vol. 70, no. 2, pp. 183–188, 2010.

[7] H. Qu, O. Mashayekhi, D. Terei, and P. Levis, "Canary: A Scheduling Architecture for High Performance Cloud Computing," arXiv:1602.01412v1 [cs.DC], 2016. [Online]. Available: http://arxiv.org/abs/1602.01412

[8] M. D. Hill and M. R. Marty, "Amdahl's Law in the multicore era," Computer, vol. 41, no. 7, pp. 33–38, 2008.

[9] H. Che and M. Nguyen, "Amdahl's Law for Multithreaded Multicore Processors," Journal of Parallel and Distributed Computing, vol. 74, no. 10, pp. 3056–3069, Oct. 2014.

[10] D. H. Woo and H. H. S. Lee, "Extending Amdahl's Law for energy-efficient computing in the many-core era," Computer, vol. 41, no. 12, pp. 24–31, 2008.

[11] S. Ristov, R. Prodan, M. Gusev, D. Petcu, and J. Barbosa, "Elastic cloud services compliance with gustafsons and amdahls laws," 2016.

[12] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing Data Transfers in Computer Clusters with Orchestra," in Proceedings of the ACM SIGCOMM 2011 Conference, ser. SIGCOMM '11, 2011, pp. 98–109.

[13] Y. Chen, R. Griffith, D. Zats, A. D. Joseph, and R. Katz, "Understanding TCP incast and its implications for big data workloads," ;login:, vol. 37, no. 3, pp. 24–38, 2012.

[14] M. Manivannan, B. Juurlink, and P. Stenstrom, "Implications of merging phases on scalability of multi-core architectures," in Proceedings of the International Conference on Parallel Processing (ICPP), 2011, pp. 622–631.

[15] M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski, "Scale-up x scale-out: A case study using nutch/lucene," in 2007 IEEE International Parallel and Distributed Processing Symposium. IEEE, 2007, p. 441.

[16] S. M. Zahedi, Q. Llull, and B. C. Lee, "Amdahl's law in the datacenter era: A market for fair processor allocation," in 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), Feb 2018, pp. 1–14.

[17] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments." in OSDI, vol. 8, no. 4, 2008, p. 7.

[18] J. Rasley, K. Karanasos, S. Kandula, R. Fonseca, M. Vojnovic, and S. Rao, "Efficient queue management for cluster scheduling," in Proceedings of the Eleventh European Conference on Computer Systems. ACM, 2016, p. 36.

[19] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on. IEEE, 2010, pp. 41–51.

[20] O. O'Malley, "Terabyte sort on apache hadoop," Yahoo, available online at: http://sortbenchmark.org/Yahoo-Hadoop.pdf,(May), pp. 1–3, 2008.

[21] "Apache hadoop," http://hadoop.apache.org/.

[22] "Ipso(beta) source code," https://github.com/ruby-/IPSO.git.

[23] M. Kleppmann, Designing Data-intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, Incorporated, 2017. [Online]. Available: https://books.google.com/books?id=X5SNswEACAAJ