5 Evaluation

They tested their system, Dolly by using a prototype build by modifying the Hadoop framework. The prototype was deployed on a 150-node cluster and evaluated using Facebook and Bing traces. In the duration of this experiment, the inter-arrival times of jobs, distribution of job sizes, and the DAG of the jobs from the original trace were preserved. The baselines for evaluating Dolly are the state-of-the-art speculation algorithms – LATE and Mantri. Here is the summary of their result:

* Average completion time of small jobs improves by 34% to 46% compared to LATE and Mantri, using fewer than 5% extra resources.
* Delay assignment outperforms CAC and CC by 2×. Its benefit increases for jobs with higher number of phases and all-to-all intermediate data flow.
* Admission control of jobs is a good approximation for preemption in favoring small jobs.

5.1 Does Dolly mitigate stragglers?

======🡺>>>>>>> insert figure 7 from anan:effective

Dolly improves the average completion time of jobs by 42\% compared to LATE and 40\% compared to Mantri, in the Facebook workload. The corresponding improvements are 27\% and 23\% in the Bing workload. Figure 7 plots the improvement in different job bins. Small jobs (bin-1) benefit the most, improving by 46\% and 37\% compared to LATE and 44\% and 34\% compared to Mantri, in the Facebook and Bing workloads. This is because of the power-law in job sizes and the policy of admission control.

========🡺>>>>>>>>> insert figure 9 from anan:effective

Figure 9 presents supporting evidence for the improvements. The ratio of medium to minimum progress rates of tasks, which is over 5 with LATE and Mantri in our deployment, drops to as low as 1.06 with Dolly. Even at the 95th percentile, this ratio is only 1.17, thereby indicating that Dolly effectively mitigates nearly all stragglers.

=======🡺>>>>>>>>>>>>insert figure 11 from anan:effective

**CC and CAC:** We now compare \cite:{anan:effective} delay assignment to the two static assignment schemes, Contention Cloning (CC) and Contention Avoidance Cloning (CAC) in Figure 11, for the Bing workload. With LATE as the baseline, CAC and CC improve the small jobs by 17\% and 26\%, in contrast to delay assignment’s 37\% improvement (or up to 2.1× better). With Mantri as the baseline, delay assignment is again up to 2.1× better. In the Facebook workload, delay assignment is at least 1.7× better. The main reason behind its better performance is its accurate estimation of the effect of contention and the likelihood of stragglers. It uses sampling from prior runs to estimate both.

5.4 Cloning Budget

===========🡺>>>>>>>>>>>insert figure 14 and 15

The improvements in the previous sections are based on a cloning budget β of 5\%. In this section, we analyze the sensitivity of Dolly’s performance to β. We aim to understand whether the gains hold for lower budgets and how much further gains are obtained at higher budgets.

In the Facebook workload, overall improvement remains at 38\% compared to LATE even with a cloning budget of only 3\% (Figure 14a). Small jobs, in fact, see a negligible drop in gains. This is due to the policy of admission control to favor small jobs. Large jobs take a non-negligible performance hit though. In fact, in the Bing workload, even the small jobs see a drop of 7\% when the budget is reduced from 5\% to 3\%. This is because job sizes in Bing are less heavy-tailed. However, the gains still stand at a significant 28\% (Figure 14b). Increasing the budget to 10\% does not help much. Most of the gains are obtained by eliminating stragglers in the smaller jobs, which do not require a big budget.

GRASS – TRIMMING STRAGGLERS

5 Implementation

GRASS was implemented on top of two data-analytics frameworks, Hadoop and Spark, representing batch jobs and interactive jobs, respectively. Hadoop jobs read data from HDFS while Spark read from in-memory RDDs. As a result, Spark tasks finish faster than Hadoop even with same sized inputs.

Implementing GRASS required two changes: task executor and job scheduler. Task executors were augmented to periodically report progress and job scheduler collects these reports, maintains samples of completed tasks and jobs, and decides the switching point.

5.1 Task Estimators

t\_rem: Tasks periodically update the scheduler with progress reportscontaining the fraction of input read and output written. Since tasks of analytics jobs are IO-intensive, we extrapolate the remaining duration of the task based on the time elapsed thus far.

t\_new: The duration of a new task is estimated by sampling from durations of completed tasks. Its value for each task is updated whenever a task completes.

**Accuracy in estimation:** While the techniques are simple, the downside is the error in estimation. According to \cite:{anan:grass}, their estimates of t\_rem and t\_new achieve moderate accuracies of 72\% and 76\%, respectively, on average.

5.2 DAG of Tasks

Jobs are typically composed as a DAG of tasks with input tasks (*e.g.,* map or extract) reading data from the underlying storage and intermediatetasks (*e.g.,* reduce or join) aggregating their outputs. Even in DAGs of tasks, the accuracy of the result is decided by the fraction of completed input tasks. ``This makes GRASS’s functioning straightforward in error-bound jobs—complete as many input tasks as required to meet the error-bound and all intermediate tasks further in the DAG \cite:{anan:grass}.``

A widely occurring property of intermediate tasks is that they perform similar functions across jobs. This idea is used for deadline-bound jobs. Also, they have relatively fewer stragglers. ``Thus, we estimate the time taken for intermediate tasks by comparing jobs of similar sizes and then subtract it to obtain the deadline for the input tasks \cite:{anan:grass}.``

6 Evaluation

GRASS was evaluated on a 200 node EC2 cluster. The results from \cite:{anan:grass} are listed below:

1. GRASS increases accuracy of deadline-bound jobs by 47% and speeds up error-bound jobs by 38%. Even non-approximation jobs (i.e., error-bound of zero) speed up by 34%. Further, GRASS nearly matches the optimal performance.

2. GRASS’s learning based approach for determining when to switch from RAS to GS is over 30% better than simple strawman techniques. Further, the use of all three guidelines discussed previously are crucial for inferring the optimal switching point.

6.1 Methodology

======🡺>>>>>>>insert table 1 here from anan:grass [simply paste the table created before]

To evaluate the performance of GRASS, they used Facebook’s production Hadoop cluster and Microsoft Bing’s production Dryad cluster. The traces capture over half a million jobs that ran across many months as shown in the table. An important thing to note here is that they did not use any approximation queries, which required them to complete all their tasks.

Job Bins: Jobs were binned by their number of tasks. They used three distinctions ``small`` (< 50 tasks), ``medium`` (51 – 500 tasks), and ``large`` (> 500 tasks).

6.2 Improvements from GRASS

6.2.1 Deadline-Bound Jobs

GRASS improves the accuracy of deadline-bound jobs by 34% to 40% in the Hadoop prototype. Gains in both the Facebook and Bing workloads are similar. Figure 5a and 5b split the gains by job size.

===============🡺>>>>>>>>>>>>>insert figure 5 from anan:grass

The gains compared to LATE as baseline are consistently higher than Mantri. Also, the gains in larger jobs are pronounced compared to small and medium jobs because their higher number of waves of tasks provide plenty of potential for GRASS. The Spark prototype improves accuracy by 43\% to 47\%. The gains are higher because Spark’s task sizes are much smaller than Hadoop’s due to in-memory inputs. Again, large jobs gain the most, improving by over 50\% (Figure 5c and 5d). better improvement of large multi-waved jobs is encouraging because smaller task sizes in future will ensure that multi-waved executions will be the norm.

6.2.2 Error-bound jobs

=========🡺>>>>>>>>>>insert figure 7

Similar to deadline-bound jobs, improvements with the Spark prototype (33% to 37%) are higher compared to the Hadoop prototype (24% to 30%). This shows that GRASS works well not only with the established frameworks like Hadoop but also upcoming ones like Spark.

6.2.3 Optimality of GRASS

Even though there were much better improvements compared to baseline, they tried to find if more improvement was possible after that. To understand the room available for improvement beyond GRASS, they compare its performance with an optimal scheduler that knows task durations and slot availabilities in advance.

======🡺>>>>>>>>>insert figure 8

Looking at this figure, we see that GRASS’s performance matches the optimal for both deadline as well as error-bound jobs. Thus, it is an efficient near-optimal solution to the NP-hard problem of scheduling tasks for approximation jobs with speculative copies.

6.3 Evaluating GRASS’s Design Decisions

6.3.1 The value of switching

============🡺>>>>>>>>>>>> insert figure 10 **AND** figure 11

To determine the importance of switching from RAS to GS, they performed experiments where they ran tasks with RAS only, GS only, and GRASS. Looking at the figure that lists their performance side-by-side, we see that GRASS’s improvements, both on average and individual bins, are strictly better than GS and RAS. To put this in numbers, its overall improvement in accuracy is over 20% better than the best of GS or RAS, demonstrating the importance of switching to reach job deadline.

6.3.2 The value of learning

It is important to know when to switch from RAS to GS without affecting the performance. There are two cases that need to be considered – static switching and adaptive switching.

**Static switching:** For deadline-bound jobs, it is the point when the time to the deadline is sufficient for at most two waves of tasks. For error-bound jobs, it is when the number of scheduled tasks sufficient to satisfy the error-bound make up two waves. GRASS’s performance is compared against strawman, which is a state-of-art baseline. Gains with the strawman are 66% and 73% of the gains with GRASS for deadline-bound and error-bound jobs, respectively. Small and medium jobs lag the most as wrong estimation of switching point affects a large fraction of their tasks. Thus, the benefit of adaptively determining the switching point is significant.

**Adaptive switching:** When only one factor is used to switch, picking either deadline or error-bound provides the best results. When two factors are used, in addition to the deadline/error-bound, cluster utilization matters more for the Hadoop prototype while estimation accuracy is important for the Spark prototype. Tasks of Hadoop jobs are longer and more sensitive to slot allocations, which is dictated by the utilization. While the smaller Spark tasks are more mutually interchangeable, this also makes them sensitive to estimation errors. Thus, in the absence of a detailed model for job execution, the three factors act as good predictors.