Evaluating CP Techniques to Plan Dynamic Resource Provisioning in Distributed Stream Processing

Andrea Reale, Paolo Bellavista, Antonio Corradi, and Michela Milano

Department of Computer Science and Engineering – DISI
Università di Bologna, Italy
{andrea.reale,paolo.bellavista,antonio.corradi,michela.milano}@unibo.it

Abstract. A growing number of applications require continuous processing of high-throughput data streams, e.g., financial analysis, network traffic monitoring, or big data analytics. Performing these analyses by using Distributed Stream Processing Systems (DSPSs) in large clusters is emerging as a promising solution to address the scalability challenges posed by these kind of scenarios. Yet, the high time-variability of stream characteristics makes it very inefficient to statically allocate the data-center resources needed to guarantee application Service Level Agreements (SLAs) and calls for original, dynamic, and adaptive resource allocation strategies. In this paper we analyze the problem of planning adaptive replication strategies for DSPS applications under the challenging assumption of minimal statistical knowledge of input characteristics. We investigate and evaluate how different CP techniques can be employed, and quantitatively show how different alternatives offer different trade-offs between problem solution time and stream processing runtime cost through experimental results over realistic testbeds.

1 Introduction

We are rapidly moving toward an always-connected world, where technology is an increasingly present mediator in the interactions between people and the environment [2]. Ever growing quantities of heterogeneous data are continuously generated and exchanged by moving or stationary sensors, smartphones, and wearable devices. This multitude of unbounded data flows must be handled effectively and efficiently.

Distributed Stream Processing Systems (DSPSs) [22] address the need of processing big data streams flexibly and in real-time by leveraging the parallel computational resources hosted inside data centers. A DSPS lets users define their own stream processing functionalities and encapsulate them in reusable components called *operators*. *Stream processing applications* (or, hereinafter, simply applications) are defined by arranging operators as vertices of *data-flow* graphs, directed and acyclic graphs that define the input-output relationships between different operators and between operators and external stream *data sources* and *data sinks*. Stream processing applications are deployed on a set of distributed resources, and their components are executed according to an event-based model that reacts to the arrival of new input data. One major problem in managing deployments of distributed stream processing applications lies in the proper management of the load fluctuations that arise due to sudden and possibly temporary variations in the rates of input data streams. If not handled properly load peaks can lead to increased processing latency due to data queuing, and to data loss due to queue overflows. To avoid these effects, it is necessary to allocate the correct amount of additional resources to overloaded applications either statically or dynamically when load variations are detected [1,5,12]. Another typical requirement is the fulfillment of *fault-tolerance* guarantees because applications usually run for (indefinitely) long time intervals and failures are unavoidable. A simple and commonly adopted solution is active replication of operator components [10], so that, if any replica fails, another can immediately take over and quickly mask the failure.

In [3,4] we have introduced LAAR, a Load Adaptive Active Replication technique that minimizes the cost of running replicated stream processing applications while guaranteeing that their deployment is never overloaded and, at the same time, that a user-specified fault-tolerance SLA is satisfied. It does so by dynamically deactivating and activating operator replicas, and by adapting the number of active ones to the changing application load according to a *replica activation strategy* precomputed before runtime. In this paper, we describe how we solve the challenging *replica activation problem*, i.e., the optimization problem whose solutions define how the LAAR runtime performs its dynamic activation of replicas depending on the currently observed configuration of input data rates. We propose a detailed study of this problem, and we present a quantitative comparison and evaluation of the effectiveness of different CP-based solution methods. The final goal is to highlight the trade-offs offered by different CP techniques considering the quality of their solutions and the associated cost.

2 Problem Definition

An application A consists of a set of components: a set I of data sources, a set P of operators, and a set O of data sinks, which collectively define the set $X = I \cup P \cup O = \{x_i\}$. The components in X are arranged in a directed acyclic application graph $W = (X, E)$. The set of edges E is described by the function:

$$pred : X \mapsto \mathcal{P}(X) \tag{1}$$

which, for each component x_i, identifies the set of predecessors $\{x_j\}$ so that $x_j \in pred(x_i) \Leftrightarrow (x_j, x_i) \in E$.

We assume that the characteristics of the application inputs are known in terms of a probability mass function that describes the probability of a source to produce data at different rates. This information could be available thanks to previous knowledge of the application domain, or inferred through an initial profiling step [9]. We also assume that the continuous space of possible rates has been properly discretized through, e.g., binning techniques [8]. In particular, every data source $x_i \in I$ can produce output at one rate among a finite set of input rates R_i. The Cartesian product $C = R_1 \times \ldots \times R_t$, where t is the number of sources, is the set of all the possible *input configurations*, and $P_C : C \mapsto [0, 1]$ is the probability mass function associated to the probability distribution of different input configurations in time. The output rate of data source $x_i \in I$ in a particular input configuration c is indicated as $\Delta(x_i, c)$.

Every operator receives one or more data streams from sources or from other operators and produces one data stream as output. For uniformity of notation, we use the symbol $\Delta(x_i, c)$ also to indicate the output rate of every operator $x_i \in P$.

Like what has been done previously in the literature (e.g., [10,11,21,20,23,25]), we summarize the characteristics of operators through a *selectivity* function δ and a *per-tuple CPU cost* function γ. For each couple (x_i, x_j) so that $x_i \in I \cup P$ and $x_j \in P$ and that $(x_i, x_j) \in E$, $\delta(x_i, x_j)$ defines the contribution of the stream generated by x_i to the output of operator x_j, so that, in absence of failures:

$$\Delta(x_j, c) = \sum_{x_i \in pred(x_j)} \delta(x_i, x_j)\, \Delta(x_i, c) \tag{2}$$

In a similar way, $\gamma(x_i, x_j)$ represents the per-tuple CPU cost for operator x_j to process tuples from x_i, so that the number of CPU cycles used by x_j per second can be expressed as:

$$\sum_{x_i \in pred(x_j)} \gamma(x_i, x_j)\, \Delta(x_i, c) \tag{3}$$

Each operator in P is actively replicated and all the replicated components are deployed on a set of distributed hosts $H = \{h_i\}$. We indicate the replicated set of operators as $\widetilde{P} = \{\tilde{x}_i^m\}$, where the symbol $\tilde{x}_i^m$ indicates the m-th replica of operator x_i. The assignment of replicas to hosts is represented by the function:

$$\vartheta : \widetilde{P} \mapsto H \tag{4}$$

For convenience, we also define $\vartheta^{-1} : H \mapsto \mathcal{P}(\widetilde{P})$ such that $\vartheta^{-1}(h) = \{\tilde{x}_i^j \in \widetilde{P} : \vartheta\left(\tilde{x}_i^j\right) = h\}$. We assume that ϑ is given, for example, because computed beforehand by an operator placement algorithm (e.g., [11,23]). In this paper, we only consider the case of twofold replication, i.e., two replicas per operator, since this scenario is the most commonly considered in real world stream applications. The model, however, can be very easily extended to k-fold replication.

A *replica activation strategy* is a function:

$$s : \widetilde{P} \times C \mapsto \{0, 1\} \tag{5}$$

that associates every operator replica – input configuration pair to one of the two possible active/inactive states. The goal of the optimization problem discussed in this paper is to find a replica activation strategy that suitably satisfies the application fault-tolerance quality requirements.

2.1 The Internal Completeness (IC) Metric

By activating/deactivating operator replicas according to the current input configuration, LAAR dynamically modifies the resilience of applications to failures. In order to measure the effect of LAAR on fault-tolerance guarantees, we define the *internal completeness* (IC) metric. Intuitively, given a failure model that describes how hosts and operators are expected to fail and a *replica activation strategy* s, the internal completeness measures, with respect to a time period T[1], the fraction of total tuples expected to be processed in case of failures compared to the number of tuples that would be processed in absence of failures.

In a no-failure scenario (best-case), the total number of tuples statistically expected to be processed by the application operators during T is:

$$BIC = T \cdot \sum_{\substack{c \in C,\\ x_i \in P,\\ x_j \in pred(x_i)}} P_C(c) \cdot \Delta(x_j, c) \tag{6}$$

Best-case internal completeness. (BIC) is the summation of the contributions of all the application operators in different input configurations, weighted by the probability of each configuration to occur.

Failure internal completeness. (FIC) measures the expected number of tuples processed with failure model ϕ and replica activation strategy s. It is defined as:

$$FIC(s) = T \cdot \sum_{\substack{c \in C,\\ x_i \in P,\\ x_j \in pred(x_i)}} P_C(c) \cdot \phi(x_i, c, s) \cdot \widehat{\Delta}(x_j, c, s) \tag{7}$$

$$\widehat{\Delta}(x_i, c, s) = \begin{cases} \Delta(x_i, c) & \text{if } x_i \in I \\ \phi(x_i, c, s) \cdot \sum\limits_{x_j \in pred(x_i)} \delta(x_j, x_i) \widehat{\Delta}(x_j, c, s) & \text{if } x_i \in P \end{cases} \tag{8}$$

The function $\phi(x_i, c, s)$ depends on the chosen failure model and describes the probability that at least one replica of operator x_i is alive and active when the input configuration is c and the replica activation strategy is s. $\widehat{\Delta}(x_i, c, s)$, instead, represents the expected output of operator x_i under failure model ϕ, when the input configuration is c and the replica activation strategy is s; note that the definition of $\widehat{\Delta}$ is recursive, as the number of tuples produced by a operator depends not only on its possible failure status (described by ϕ) but also on the number of tuples produced by its predecessor (8). Let us rapidly note that the possible failures of data sources, which are components external to the application, are assumed to be handled externally.

Internal completeness (IC) is defined as the ratio between FIC and BIC:

$$IC(s) = \frac{FIC(s)}{BIC} \tag{9}$$

[1] We choose T long enough for the statistical characteristics of the sources to apply.

2.2 The Replica Activation Problem

In this section, we define the optimization problem that, solved off-line and before deployment time, outputs a replica activation strategy that fits the application fault tolerance requirements, and that is used at runtime by LAAR to activate operator replicas.

We call this problem *replica activation problem*, and we define it as follows:

$$\underset{s}{\text{minimize}} \quad \mathit{cost}\,(s) \tag{10a}$$

subject to:

$$IC(s) \geq G \tag{10b}$$

$$\sum_{\substack{\tilde{x}_i^m \in \vartheta^{-1}(h),\\ x_j \in pred(x_i)}} \gamma\,(x_j, x_i)\,\Delta(x_j, c) s(\tilde{x}_i^m, c) \leq K_h \qquad \substack{\forall h \in H,\\ \forall c \in C} \tag{10c}$$

$$s\left(\tilde{x}_i^0, c\right) + s\left(\tilde{x}_i^1, c\right) \geq 1 \qquad \substack{\forall x_i \in P,\\ \forall c \in C} \tag{10d}$$

The *cost* function in the minimization term represents the cost, in terms of resources, for a service provider to run the application using replica activation strategy s and the replicated assignment defined by ϑ. In this work, we assume the bandwidth available for cluster-local communication to be an abundant resource (a common assumption in data center contexts), and we model our cost function as the total number of CPU cycles used in a period T. It is defined as follows:

$$\mathit{cost}\,(s) = T \sum_{\substack{c \in C,\\ \tilde{x}_i^m \in \tilde{P},\\ x_j \in pred(x_i)}} P_C\,(c)\,\gamma\,(x_j, x_i)\,\Delta(x_j, c) s(\tilde{x}_i^m, c) \tag{11}$$

and is the summation over the CPU consumption of all the operator replicas.

Equation (10b) constrains IC to satisfy a requested fault-tolerance value G, while (10c) states that each host in the deployment should never be overloaded; K_h is a constant expressing the number of CPU cycles per second available at host h. The last constraint, expressed in (10d), requires that there is at least one active replica of every operator in every input configuration, and it ensures that the measured IC value is one in absence of failures.

2.3 Failure Model

We consider a simplified failure model ϕ, based on the following assumptions:

1. In any failure scenario, one replica of every operator fails.
2. Unless both the replicas are active at some point in time, the non-failed replica is assumed to be the one that was inactive according to the replica activation strategy.
3. Once failed, replicas never recover.

or, more formally:

$$\phi(x_i, c, s) = \begin{cases} 0 & \text{if } s(\tilde{x}_i^0, c) + s(\tilde{x}_i^1, c) < 2,\ \tilde{x}_{i,l} \in \widetilde{P} \\ 1 & \text{otherwise} \end{cases} \tag{12}$$

This model will in general overestimate possible failure conditions because, in the actual stream processing deployment, it is highly unlikely that replicas of every operator fail at the same time, and because, in our runtime LAAR implementation, when an operator failure is detected, any corresponding and possibly deactivated replica returns to its active state, while an automatic recovery procedure promptly replaces the failed component [4]. For these reasons, we refer to ϕ as *pessimistic failure model.* While, overstating, on the one hand, the effects and consequences of failures compared to actual runtime conditions, on the other hand, this choice of ϕ provides two fundamental benefits: (i) the IC value computed using this model is a large lower bound to the real IC that will be observed on actual application deployments because any real failure condition is highly likely to be much less severe than those predicted by the model; (ii) its mathematical formulation simplifies the computation of IC values for different possible replica activation strategies and hence the optimization complexity.

Note that the solution space of this problem is still very large, as for every application there are $3^{|P|\cdot|C|}$ possible replica activation strategies. Note also that, in cost function (11), the IC constraint (10b) and the hosts CPU constraints (10c) depend on $\widehat{\Delta}(x_i, c, s)$ (8), which is a recursively defined exponential term. Hence, to find algorithms that can find optimal or good enough solutions to this problem is a major technical challenge.

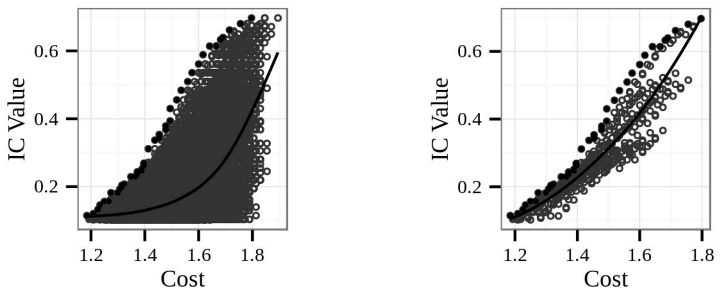
3 Solving the Problem with CP

In this section, we analyze different CP-based approaches to solve the replica activation problem presented in Section 2.

As a first straightforward solution, we implemented the optimization model (10) "as-is" on the commercial IBM ILOG CP Optimizer solver [13], and used it to get a better understanding of the problem structure.

Looking at the problem from a user perspective, cost (11) and IC (9) are the most important parameters because, together, they determine the cost-quality trade-off for running stream processing applications with LAAR. Intuitively, since the basic mechanism to mask the effects of failures is to activate more replicas, in general, requiring higher IC (and hence better fault-tolerance) will correspond to higher runtime costs.

Fig. 1a gives some insight about the shape of the problem solution space when considering together cost and IC: it shows the space of possible feasible solutions of a problem instance consisting of 24 operator replicas distributed on 6 computing hosts and IC constraint — G in (10b) — set to 0.1. The continuous black line is a *loess* regression [6] of the solution points and confirms that, as a general trend, the cost of solutions is proportional to their IC value. However, the graph also shows



(a) (b)

Fig. 1. Cost–IC relationship in the solution space of a problem instance consisting of 12 operators (2 replicas each) deployed on 6 hosts. Full circles represent the Pareto frontier of the problem space, while the continuous line is a regression of the solution points. Without (a) and with partial filtering of sub-optimal solutions (b).

that there is a very large number of sub-optimal solutions (empty circles) and that higher costs do not necessarily imply higher IC. Recall that the IC value does not only depend on the number of active replicas, but also on the particular choice of operators to activate and on the topology of the application data flow graph. As a consequence, a wrong choice of active operator replicas can easily lead to a useless waste of resources.

However, an important fraction of sub-optimal solutions not belonging to the Pareto frontier of the solution space can be discarded quickly with simple considerations. For example, think about a pipeline of operators where a first operator (O0) feeds a second one (O1). Given the pessimistic failure model in (12), having, in any input configuration, two active replicas of O1 and, at the same time, only one active replica for O0 does not contribute to the overall IC value because, in case of failures, O1 would not receive any sample to process from O0; that would, however, increase the solution cost. This is not only valid for pipelines but can be generalized for any graph shape: in particular, any feasible replica activation strategy s_x that, in some input configuration c, has two active replicas for some operator x_i whose predecessors all have only one active replica is sub-optimal with respect to a corresponding feasible replica activation strategy s_y that differs from s_x only for the fact that x_i has just one active replica. This relation can be used to add a new constraint to (10) that approximates the set of Pareto-optimal points by performing a *partial sub-optimal solution filtering* (PSF), which removes obviously sub-optimal solutions. We formulate this constraint as follows:

$$\exists\, x_i \in P, c \in C \text{ s.t. } \forall x_j \in \mathit{pred}\,(x) \sum_{l=0,1} s\left(\tilde{x}_j^m, c\right) = 1 \implies \sum_{l=0,1} s\left(\tilde{x}_i^m, c\right) = 1 \tag{13}$$

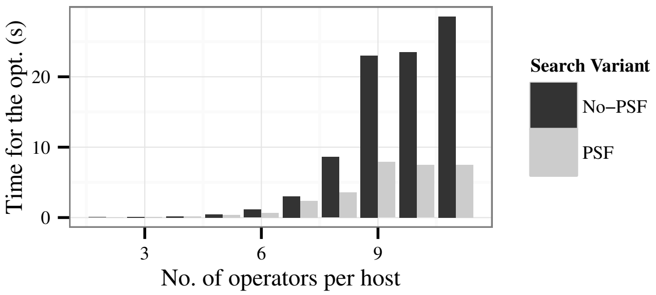


Fig. 2. Comparison of average time to find an optimal solution with and without the partial sub-optimal solution filtering

Fig. 1b shows the solution space of the same problem instance in Fig. 1a after the filtering based on (13). This important reduction in size also has a significant impact on the time needed to solve the problem. Fig. 2 summarizes the average search time needed to find the optimum solutions for a batch of small problem instances in which graphs of 2 to 11 operators are deployed on 4 hosts with two replicas per operator. As the graph complexity increases, the benefit of the additional constraint in (13) becomes more and more evident.

Given these characteristics of the solution space, we have developed three search strategies. The first (called *Basic*) is the straightforward implementation of the model in (10) with the additional constraint in (13) on the ILOG CP Optimizer solver. Since the realization of this first strategy is straightforward, we do not detail it further in this paper. On the contrary, in the following two subsections, we introduce the second and third search strategies, respectively a *Large Neighborhood Search (LNS)-based strategy* and a *decomposition-based* one.

3.1 LNS-Based Strategy

The basic idea behind LNS strategies [19] is to start from an initial solution and then proceed through incremental improvement steps that focus on *large neighborhoods* of the current best solution. We developed a strategy to solve our replica activation problem that is based on these concepts. The algorithm starts from a solution found either by using the *Basic* solver presented in the previous section, or by leveraging a simple *greedy* initial solution. The algorithm that finds this last type of initial solution is very simple: it starts with a replica activation strategy where two replicas of every operator are always active; iteratively, it deactivates the most resource hungry operator until all the non-overloading condition constraints (10c) are met. The advantage of using this greedy algorithm to find the initial solution four our LNS approach is its ability to terminate extremely quickly. These greedy solutions are not necessarily feasible because they can violate the constraint (10b), but our practical experience has shown that those infeasibilities can be often rapidly corrected through a few LNS moves.

Given the initial problem solution, the LNS-based strategy proceeds through a series of iterative improvement steps. At every round, a new optimization problem is built by *relaxing* the current best solution, i.e., by fixing the values of a subset of the search variables to those of the best solution so far and by focusing the exploration on the subspace of the remaining *relaxed* variables. We choose the variables to relax at each iteration according to one of two alternative methods. In the first (*simple random*), they are chosen completely random; in the second (*weighted random*), every search variable is assigned a weight that depends on its corresponding input configuration, so that variables associated to more resource-hungry input configurations (typically corresponding to load peaks) have higher chances to be chosen for relaxation. The weighted random strategy aims at relaxing these variables first because they usually require the highest number of operator replicas deactivated in order to satisfy (10c) and consequently have a stronger influence on the satisfiability of the IC requirement (10b). Every iterative improvement step explores the corresponding relaxed subspace either until it finds a solution that improves upon the previous one or until a local time limit expires. Every round dynamically adapts the number of relaxed variables by reducing it when the last step has produced improvements or by increasing it in case of deteriorations of the solution quality. The algorithm terminates either when there is no improvement for a configurable number of consecutive rounds or when a global time limit expires.

Note that, differently from the Basic search strategy, the LNS-based one does not recognize when an optimum has been reached, and, when greedy starting points are used, it cannot conclude whether a problem has any solutions if none are found.

3.2 Decomposition-Based Strategy

In this section we propose a solution approach that decomposes the problem in a number of orthogonal subproblems along its $|C|$ different input configurations. The goal of this decomposition-based approach is to provide scalability especially for instances with a large number of input configurations. This type of optimization is very important in many common real-world problems, where stream processing applications process data from tens of sources, each producing data at different possible data rates.

Let us consider once again the formulation of the replica activation problem (10). Separating the CPU constraints in (10c) and the minimum replicas constraints in (10d) is trivial, because each of them involves only terms relative to a single input configuration c. The search variables s can be equally easily separated by considering $|C|$ different replica activation strategies s_c such that:

$$s(\tilde{x}_i^m, c) = s_c(\tilde{x}_i^m), s_c : \widetilde{P} \mapsto \{0,1\} \tag{14}$$

The IC constraint (10b) can be, instead, rewritten as follows:

$$
\begin{aligned}
&\frac{FIC(s)}{T} \geq \underbrace{\frac{BIC}{T}(G)}_{G'} \\
&\Leftrightarrow \sum_{c \in C} \underbrace{\sum_{\substack{x_i \in P, \\ x_j \in pred(x_i)}} P_C(c) \cdot \phi(x_i, c, s) \cdot \widehat{\Delta}(x_j, c, s)}_{\mu_c(s_c)} \geq G' \\
&\Leftrightarrow \sum_{c \in C} \mu_c(s_c) \geq G' \qquad (15)
\end{aligned}
$$

Similarly, considering (11), the minimization term (10a) can be written as:

$$
\begin{aligned}
&\min \sum_{c \in C} \underbrace{\sum_{\substack{\tilde{x}_i^m \in \tilde{P}, \\ x_j \in pred(x_i)}} P_C(c)\, \gamma(x_j, x_i)\, \Delta(x_j, c) s(\tilde{x}_i^m, c)}_{\lambda_c(s_c)} \\
&\Leftrightarrow \min \sum_{c \in C} \lambda_c(s_c) \qquad (16)
\end{aligned}
$$

Note that, while the CPU and minimum replicas constraints can be evaluated and satisfied considering each input configuration c separately, the IC constraint and the cost minimization expression cannot; nonetheless, they both can be expressed as a sum of $|C|$ non negative terms, and each of this terms can be evaluated separately for different values of c.

Our decomposition approach consists in defining $|C|$ subproblems $prob_c$, one per input configuration; the solution of each problem is a partial replica activation strategy s_c that satisfies at least the corresponding CPU and minimum replication constraints (10c) and (10d). The subproblems' optimization goal and possible additional constraints, instead, depend on the particular phase the decomposition algorithm is in. Algorithm 1 sketches, in pseudo-code, the main steps of the decomposition-based solver.

The algorithm starts by maximizing the $\mu_c(s_c)$ values of each subproblem (Phase 1, lines 1–9). Note that, after this phase is complete, if a solution is found for every subproblem, an upper bound on the possible IC for the original problem can be obtained using (15): through it, it is possible to test immediately whether the original problem admits solutions (line 7) and, in case it does, to output an initial and in general sub-optimal solution. During Phase 2 (lines 10–22), this initial solution is improved by working separately and iteratively on each subproblem. At every iteration, the problem whose contribution to the overall IC is minimum with respect to its contribution to the cost (line 12) is chosen as a candidate for improvement, and the algorithm tries to decrease its cost while ensuring that the obtained $\mu_c(s_c)$ value still allows to satisfies the overall IC requirement (line 13). This iteration is repeated until no improvement

Algorithm 1. Decomposition-based Search Strategy

```
input  : $\{prob_c\}$: the $|C|$ decomposed subproblems.
output: A replica activation strategy $s$, or None if no solution found
Phase 1:                                                    /* $\mu_c$ maximization */
  foreach $prob_c$ do
      $s_c^{\max} \leftarrow$ maximize $\mu_c$ in $prob_c$
      if $s_c^{max}$ is None then return None $\mu_c^{\max} \leftarrow$ maximum $\mu_c$ for $prob_c$
      $\lambda_c^{\max} \leftarrow$ cost value corresponding to $s_c^{\max}$
  end
  if $\sum_{c \in C} \mu_c^{max} < G'$ then                  /* Feasibility test */
      return None
  end
Phase 2:                                                    /* optimization */
  foreach $prob_c$ do $\mu_c^{\mathrm{cur}} \leftarrow \mu_c^{\max}$; $\lambda_c^{\mathrm{cur}} \leftarrow \lambda_c^{\max}$ while exists $prob_c$ that can be
  improved do
      $c' \leftarrow \max_c (\lambda_c^{\mathrm{cur}}/\mu_c^{\mathrm{cur}})$           /* Choose prob. to improve */
      $\mu_{c'}^{\mathrm{limit}} \leftarrow G' - \sum_{\substack{c \in C \\ c \neq c'}} \mu_c^{\mathrm{cur}}$
      Post $\mu_{c'} \geq \mu_{c'}^{\mathrm{limit}}$ as constraint on $prob_{c'}$
      Post $\lambda_{c'} < \lambda_{c'}^{\mathrm{cur}}$ as constraint on $prob_{c'}$
      $s_{c'}^{\mathrm{cur}} \leftarrow$ findFirst($prob_{c'}$)                  /* Solve $prob_c$ */
      if $s_{c'}$ is None then
          $prob_{c'}$ cannot be improved further
      else
          Update $\mu_{c'}^{\mathrm{cur}}$ and $\lambda_{c'}^{\mathrm{cur}}$ according to $s_{c'}^{\mathrm{cur}}$
      end
  end
Phase 3:                                                    /* End */
  $s \leftarrow$ Combine all the $s_c^{\mathrm{cur}}$
  return $s$
```

can be obtained from any subproblem. In Phase 3, finally, the partial replica activation strategies are combined, and the result returned as output.

Like the LNS-based strategy, this algorithm can decide whether the problem is feasible, but cannot recognize an optimal solution. In the cases where the operator graph is particularly complex, it might be necessary to set a time limit for Phase 1 to avoid blocking the solver for too long; in such cases, the solution obtained after Phase 1 is no longer an upper bound on the obtainable IC, and so the algorithm cannot decide anymore about the feasibility of the entire problem. Let us note, finally, that the various subproblems optimizations (either the initial IC maximization or the subsequent cost minimizations) can be performed with any optimization technique.

4 Experimental Evaluation

The primary goals of our evaluation study are i) to compare the quality of the best solutions that the three strategies can find within a reasonable time limit and ii) to evaluate the scalability of the search strategies (in particular of the decomposition-based one) as the problem size grows.

For the first part of this evaluation, we consider a batch of 20 different stream processing applications with data flow graphs of 96 operators each. Every application has three data sources, each producing output at two possible data rates (for a total of 8 input configurations), and is associated with a replicated deployment (two replicas per operator, 192 replicas in total) on 24 computing hosts. The IC constraint in the related Replica Activation Problem is set to 0.5. We choose these applications as we believe their complexity to be well representative of real world stream processing deployments.

We compare the optimization algorithms in the following variants:

1. Basic solver with partial sub-optimal-filtering (*BASIC*).
2. LNS-based strategy using BASIC for the initial solution and simple weighted random relaxation method (*L_SRW*).
3. LNS-based strategy using a greedy initial solution and weighted random relaxation method (*L_GRW*).
4. LNS-based strategy using a greedy initial solution and simple random relaxation method (*L_GRS*).
5. Decomposition strategy using BASIC for Phase 1 (*DEC_S*).
6. Decomposition strategy using LNS_GRW for Phase 1 (*DEC_L*).

All the experiments are executed on a machine with an AMD Phenom II X6 1055T @2.8 GHz processor and 8 GB of main memory. The ILOG CP Optimizer is configured to use only one worker (single threaded solution), and its search time limit is set to 300 seconds wall time. Due to the complexity of the problems, for no instance it was possible to demonstrate the optimality of the solutions found; however, feasible solutions were found for all instances except four.

In this experimental campaign, we were primarily interested in two aspects, i.e., the ability to find good solution in a relatively large time frame, and the complementary capacity of quickly finding a first feasible solution. Both aspects are critical in our use-case scenario: the first is more significant during the deployment of new stream processing applications, when a larger time budget is usually available; the second is more relevant when replica activation strategies must be quickly adjusted at runtime due to dynamic variations of input characteristics. For reasons of space, in the following, we only report the results about the first aspect. Experimental data about the second are available in our on-line appendix [17], together with downloadable descriptions of the problem instances used in this evaluation.

The bar plot in Fig. 3 compares the various search algorithms to BASIC, which we choose as the base line solution method; the plot analyzes the best solution cost (BCOST) and its associated search time (BTIME) and is obtained by normalizing, separately for each problem instance, the values of BTIME and BCOST with respect to the results obtained in BASIC. The figure shows the average of these values along with the associated standard error.

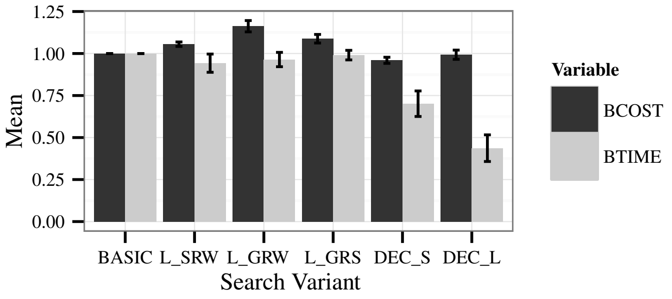


Fig. 3. Mean time to find the best solution within the time limit and associated solution cost. All the results are normalized w.r.t. the BASIC variant.

In general, the two decomposition-based variants find solutions that are at least as good or slightly better than those found by BASIC. In more detail, in the only six instances where the decomposition variants finds solutions worst than BASIC, that solution is at most 23% more expensive; at the same time, they can save considerable amounts of time (43% on average). The results show also that, on the one hand, DEC_L can find good solutions much faster than DEC_S (4% to 70% faster), probably due to the initial speed-up given by the LNS greedy strategy used to find the starting solutions for Phase 1; on the other hand, the solutions found by DEC_L tend to be a little more expensive than those found by DEC_S (from 1% to 24%): that is explained by considering that the use of the BASIC solver in Phase 1 usually gives tighter IC upper bounds, which, in turn, permit to use looser constraints on the μ_c^{limit} values in Phase 2. The solutions found by the LNS-based variants, finally, are in all but one case worst than those found by BASIC, with cost inflations up to 57%. Among the LNS variants, L_SRW is the one providing the best results thanks to its better (although slower) initial solutions, with higher costs (between 1% and 13%), but solution times that are 75% smaller to 26% bigger than BASIC.

Finally, we evaluate the scalability of the decomposition-based strategy when the number of input configurations grows. In order to measure it, we started from an application graph with 32 operators and one data source, with a replicated deployment (64 operators) on 8 hosts, and we randomly generated 40 different applications, for each, customizing the number of possible data source rates. The result is a set of 40 different replica activation problem instances sharing the same processing graph and deployment, but with a progressively growing number of input configurations (from 2 to 80 by steps of 2). We solved these instances through the BASIC and the DEC_S search variants. Fig. 4 shows the time taken by the two strategies to find their best solution as the number of input configurations grows. The results for the BASIC variants grow very quickly, and, for instances with more than 18 configurations, we could not find any solution within the time limit. On the contrary, by using DEC_S, the solution time grows much more slowly, and we easily solved all the problem instances.

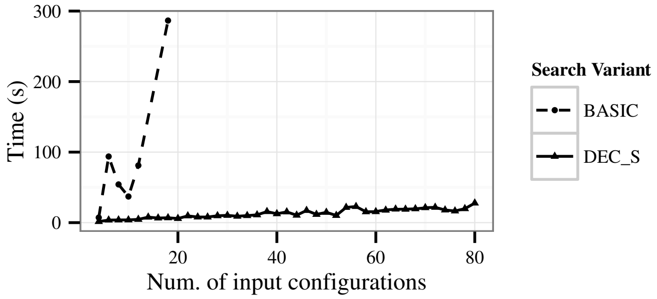


Fig. 4. Scalability of the BASIC and DEC_S solution strategies

5 Related Work

The problem of managing load variations in DSPSs has been widely investigated in the related literature, and different solution techniques have been proposed.

Tatbul et al. [20], for example, tries to avoid resources overload in spite of changing load condition by introducing controlled data drops in the data-flow graph. In a pre-deployment phase, they solve a set of LP problems to build load-shedding plans that decide where and how many tuples to drop to maximize application throughput. Like our *replica activation strategies*, load-shedding plans contain decisions for each input configuration. However, these plans do not take into account the interplay of fault-tolerance and variable input load: with load shedding, data is dropped even when no failure occurs, while LAAR guarantees that no data is lost in that case and it bounds the maximum amount of loss in case of failures. For these reasons, the problem model in [20] and the one presented in this paper are very different and difficult to compare.

Another common approach is to move operators between hosts to re-balance the system and accommodate new load conditions: in [12,24,25] this is done through continuous greedy improvement steps. Likewise, in [1], the authors develop a resource allocation algorithm that uses a dynamic flow-control algorithm based on a linear quadratic regulator [7] to maximize the application throughput. All these approaches assume that the available resources are enough to handle any input configuration, or that the data sources rate can be paced at will until there are enough resources to handle the load; in LAAR additional resources are dynamically provided by temporarily replication adjustments.

Using CP to manage replicas in distributed systems has been previously done by Michel et al. [16], who propose a CP model that solves the problem of deploying replicas on distributed nodes to minimize the communication cost in Eventually Serializable Data Service (ESDS) systems. Our replica activation problem is different because we do not deal with the assignment of replicas to computing resources, but we decide their dynamic activation strategy.

In [15], the authors solve a combined assignment and scheduling problem for conditional task graphs (CTG). Similarly to this work, the CP model includes stochastic elements, but they are used to describe the probability that branches in the task graphs are actually used at runtime.

Our problem formulation closely resembles stochastic optimization problems with value at risk (VaR) guarantees [18], and our decomposition strategy is based on the notion of separability of optimization problems commonly used in OR contexts [14].

6 Conclusions and Future Work

In this paper we have introduced an optimization problem whose solution is at the foundations of LAAR, a technique for dynamic and load-adaptive active replication in DSPSs. After having investigated the characteristics of the problem and of its solution space, we have presented three possible solution strategies: a naïve and straightforward model solver, a LNS-based search strategy, and a Decomposition-based strategy. Our experimental evaluation shows that when sufficient time budged is available, the decomposition approach can find good solutions and scale particularly well for instances where possibly many data sources produce input at many possible rates. On the contrary, the LNS-approaches represent an appropriate solution when finding quickly a good-enough feasible solution is the main concern [17]. Finally, the naïve solver provides the most consistent behavior across all the possible scenarios when used in combination with an ad-hoc sub-optimal solution filtering constraint.

As future work, we will continue our investigation on the problem trying to correlate specific problem characteristics (e.g., shape of the graph, properties of the deployment) to the behavior of different search strategies. In addition, we will continue experimenting with the current approaches on a broader set of problem instances in order to expand and further validate our findings. Let us finally note that, although we have introduced the replica activation problem and LAAR in the context of stream processing, the presented principles are applicable to the much larger domain of distributed data flow systems that can tolerate weaker fault-tolerance levels through dynamic active replication.

Acknowledgements. We would like to thank the Smarter Cities Technology Centre, IBM Dublin Research Laboratory and, in particular, Spyros Kotoulas for the precious contribution in the development of the original ideas behind LAAR.

References

1. Amini, L., Jain, N., Sehgal, A., Silber, J., Verscheure, O.: Adaptive control of extreme-scale stream processing systems. In: Proc. of the 26th IEEE ICDS Conference, pp. 71–78. IEEE (2006)
2. Atzori, L., Iera, A., Morabito, G.: The internet of things: A survey. Computer Networks 54(15), 2787–2805 (2010)

3. Bellavista, P., Corradi, A., Kotoulas, S., Reale, A.: Dynamic datacenter resource provisioning for high-performance distributed stream processing with adaptive fault-tolerance. In: Proc. of the 2013 ACM/IFIP/USENIX International Middleware Conference. Posters and Demos Track (2013)
4. Bellavista, P., Corradi, A., Kotoulas, S., Reale, A.: Adaptive fault-tolerance for dynamic resource provisioning in distributed stream processing systems. In: Proc. of the of 17th International EDBT Conference. ACM (2014)
5. Boutsis, I., Kalogeraki, V.: Radar: adaptive rate allocation in distributed stream processing systems under bursty workloads. In: Proc. of the 31st SRDS Symposium, pp. 285–290. IEEE (2012)
6. Cleveland, W.S., Devlin, S.J.: Locally weighted regression: an approach to regression analysis by local fitting. J. Amer. Statist. Assoc. 83(403), 596–610 (1988)
7. Cobb, D.: Descriptor variable systems and optimal state regulation. IEEE Transactions on Automatic Control 28(5), 601–611 (1983)
8. Dougherty, J., Kohavi, R., Sahami, M.: Supervised and unsupervised discretization of continuous features. In: Proc. of the 12th ICML Conference, pp. 194–202. Morgan Kaufmann (1995)
9. Gedik, B., Andrade, H., Wu, K.-L.: A code generation approach to optimizing high-performance distributed data stream processing. In: Proc. of the 18th CIKM Conference, pp. 847–856. ACM (2009)
10. Hwang, J.-H., Balazinska, M., Rasin, A., Çetintemel, U., Stonebraker, M., Zdonik, S.: High-availability algorithms for distributed stream processing. In: Proc. of the 21st ICDE Conference, pp. 779–790. IEEE (2005)
11. Khandekar, R., Hildrum, K., Parekh, S., Rajan, D., Wolf, J., Wu, K.-L., Andrade, H., Gedik, B.: Cola: Optimizing stream processing applications via graph partitioning. In: Bacon, J.M., Cooper, B.F. (eds.) Middleware 2009. LNCS, vol. 5896, pp. 308–327. Springer, Heidelberg (2009)
12. Kumar, V., Cooper, B., Schwan, K.: Distributed stream management using utility-driven self-adaptive middleware. In: Proc. of the 2nd ICAC Conference, pp. 3–14. IEEE (2005)
13. Laborie, P.: Ibm ilog cp optimizer for detailed scheduling illustrated on three problems. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 148–162. Springer, Heidelberg (2009)
14. Li, D., Sun, X.: Separable integer programming. In: Nonlinear Integer Programming, ch. 7, pp. 209–239. Springer (2006)
15. Lombardi, M., Milano, M.: Allocation and scheduling of conditional task graphs. Artificial Intelligence 174(78), 500–529 (2010)
16. Michel, L., Shvartsman, A.A., Sonderegger, E., Van Hentenryck, P.: Optimal deployment of eventually-serializable data services. In: Perron, L., Trick, M.A. (eds.) CPAIOR 2008. LNCS, vol. 5015, pp. 188–202. Springer, Heidelberg (2008)
17. Reale, A., Bellavista, P., Corradi, A., Milano, M.: Evaluationg cp techniques to plan dynamic resource provisioning in distributed stream processing: On-line appendix, `http://middleware.unibo.it/people/ar/laar-rap/` (web page, last visited in February 2014)
18. Rockafellar, R.T., Uryasev, S.: Optimization of conditional value-at-risk. Journal of Risk 2, 21–42 (2000)
19. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M.J., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998)

20. Tatbul, N., Çetintemel, U., Zdonik, S.: Staying fit: efficient load shedding techniques for distributed stream processing. In: Proc. of the 33rd VLDB Conference. The VLDB Endowment (2007)
21. Tatbul, N., Çetintemel, U., Zdonik, S., Cherniacak, M., Stonebraker, M.: Load shedding in a data stream manager. In: Proc. of the 29th VLDB Conference, pp. 309–320. The VLDB Endowment (2003)
22. Turaga, D., Andrade, H., Gedik, B., Venkatramani, C., Verscheure, O., Harris, J., Cox, J., Szewczyk, W., Jones, P.: Design principles for developing stream processing applications. Soft. Pract. Exper. 40(12), 1073–1104 (2010)
23. Xing, Y., Hwang, J.-H., Çetintemel, U., Zdonik, S.: Providing resiliency to load variations in distributed stream processing. In: Proc. of the 32nd VLDB Conference. The VLDB Endowment (2006)
24. Xing, Y., Zdonik, S., Hwang, J.H.: Dynamic load distribution in the borealis stream processor. In: Proc. of the 21st ICDE Conference, pp. 791–802. IEEE (2005)
25. Zhou, Y., Ooi, B.C., Tan, K.-L., Wu, J.: Efficient dynamic operator placement in a locally distributed continuous query system. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 54–71. Springer, Heidelberg (2006)