Transactions on Environment, Energy and Earth Sciences
ISSN: 2960-1924, eISSN: 2960-2300 | Volume 7
GTREE 2026

Performance-Aware Scheduling and Resource Allocation for Large-Scale Structural Simulations

Dai Teng

University of Illinois Urbana-Champaign, Urbana, IL, USA

tengdai@ieee.org

Abstract. Seismic performance assessment requires thousands of nonlinear response-history analyses, typically organized as incremental dynamic analysis (IDA) campaigns that share clusters with other users. These campaigns differ from generic batch workloads: every ground motion induces a chain of analyses whose next intensity level is needed only if the current one does not collapse, runtimes are governed by record length, model cost and intensity, and memory requests are sized for the worst case. We measured 810 real OpenSees analyses of four archetype frames under the 22 FEMA P695 far-field records and show that chain parallelism decays sharply as records collapse, that history-based runtime predictors are uncorrelated with actual runtimes, and that static memory requests exceed footprints by a median factor of 2.28. We propose PASS, a cross-task scheduler that combines an intensity-conditioned runtime and memory model, an online collapse-hazard estimator yielding the survival-weighted remaining work of every chain in constant time, a bound-guided critical-chain priority with aged shortest-job-first ordering, probability-gated and preemptible speculative execution of future intensity levels on idle cores, and memory right-sizing with out-of-memory feedback. In trace-driven experiments against FIFO, EASY backfilling, shortest-job-first with history-based and exact runtimes, static partitioning, Tetris and critical-path-first scheduling, PASS shortens the makespan of IDA batches by 33.9% and reduces the mean waiting time of online campaigns by 89.6% relative to the best baseline, raises core utilization from at most 57.8% to 74.6–79.5%, and requires about 142μs per scheduling decision. On 1024-core clusters, where every baseline is bounded by the longest chain, PASS reduces the makespan by 56%. Ablation, sensitivity and robustness experiments quantify the contribution of each component and show that the gains persist under runtime noise, heterogeneous nodes and bursty arrivals.

Keywords: Structural simulation; Incremental dynamic analysis; OpenSees; Job scheduling; Resource allocation; Speculative execution; High-throughput computing.

1. Introduction

Performance-based earthquake engineering has turned nonlinear response-history analysis into a high-throughput computing problem. A collapse assessment that follows the FEMA P695 methodology [1] subjects every structural archetype to incremental dynamic analysis (IDA) [2], [3] under a suite of ground motions, each scaled to a ladder of intensity levels until sidesway collapse. A design study with a dozen archetypes therefore requires tens of thousands of OpenSees [4], [5] analyses, and parametric, uncertainty-propagation and regional studies multiply this count: fault-to-structure frameworks now couple regional ground-motion and structural-response simulations [6], and community platforms orchestrate building-level analyses for entire regions [7]. Such campaigns are executed on shared clusters and science gateways, where they compete with the work of other users for cores and memory.

The execution of these campaigns is normally delegated to general-purpose batch policies—first-come-first-served (FCFS) with backfilling [8] as provided by resource managers such as SLURM [9]—or to static per-group partitions of a cluster. OpenSees already offers parallel interpreters for domain decomposition and parameter sweeps, and task-farming has been used to distribute a single IDA study over networked processors [10]. These mechanisms accelerate one analysis or distribute a fixed list of runs; they do not decide which analysis of which campaign should occupy a free core

Content from this work may be used under the terms of CC BY-NC 4.0 licence (https://creativecommons.org/licenses/by-nc/4.0/).
Published by Warwick Evans Publishing.

next, nor how much memory to reserve for it. We identify three properties of IDA campaigns that make generic policies inefficient.

(P1) Conditional dependencies. In the stepping and hunt-and-fill tracing algorithms [3], the run at intensity level $k+1$ of a record is needed only if the run at level k did not collapse. The task graph is a set of chains whose lengths are revealed only at run time. The parallelism of a campaign therefore decays as records reach collapse, and the last surviving chains dominate the makespan. A list scheduler cannot shorten this tail, because from its point of view the next level of a chain does not yet exist.

(P2) Structured, partially predictable cost. The runtime of a response-history analysis is governed by the number of integration steps (known from the record length before execution), the per-step cost of the model (observable after the first runs of a campaign) and the degree of nonlinearity, which grows with intensity and is truncated when collapse terminates the analysis early. History-based predictors designed for generic jobs [11] ignore this structure.

(P3) Static, worst-case memory requests. Users size the memory request of a campaign for its largest run, whereas the footprint of most runs is smaller and predictable from the model and the output volume. On memory-constrained nodes the unused reservation strands cores, a pattern also reported for production clusters [12] and scientific workflows [13], [14], [15].

This paper presents Pass (Performance-Aware Structural-simulation Scheduler), a cross-task scheduler and resource allocator for large-scale structural simulation campaigns that exploits P1–P3. Its design follows the uncertainty-aware predictive scheduling principle established by PRESAGE [16] for elastic cloud analytics and inference services: predictive uncertainty is not reported as a separate, descriptive output but converted directly into allocation decisions. In Pass, the uncertain quantities are whether a future IDA level will be needed at all and what it will cost, and the decisions are which run to start, how much memory to reserve and which idle cores to fill with speculative runs. The contributions are as follows.

Measurement-based workload characterization. We execute 810 real OpenSees response-history analyses (1.00 CPU-hours) of four archetypes—three steel special moment frames with deteriorating concentrated plasticity and one reinforced-concrete frame with fiber force-based elements—under the 22 FEMA P695 far-field records, and quantify runtime and memory heterogeneity, the distribution of collapse levels and the decay of chain parallelism (Section 3).

Intensity-conditioned estimation coupled with an online collapse-hazard model. Runtime is modeled as planned steps $\times$ per-step cost $\times$ intensity growth, mixed over the collapse and non-collapse outcomes. A discrete collapse hazard is estimated per campaign by Beta–binomial shrinkage toward a lognormal fragility prior, which yields the survival-weighted remaining work of every chain in $O(1)$ time through suffix sums.

A bound-guided priority rule. A chain is treated as critical when its expected remaining work exceeds a fraction θ of the per-core share of all outstanding work, i.e., when the chain rather than the aggregate load controls the makespan lower bound. Critical chains are served longest-first, all other tasks by aged shortest-expected-job-first.

Probability-gated, preemptible speculative execution. Idle cores execute future intensity levels whose probability of being needed exceeds $p_{\min}$; speculative runs are cancelled when a predecessor collapses and are preempted whenever confirmed work needs the core. We prove that confirmed work is never delayed and that chain progress is accelerated by up to $(D+1)\times$ for speculation depth D.

Memory right-sizing with out-of-memory (OOM) feedback, which reserves the predicted footprint plus an adaptive margin and falls back to the user request after a kill.

Extensive evaluation. Trace-driven simulation of 64–1024-core clusters against seven baselines—FIFO, EASY backfilling, SJF with history-based estimates, SJF with exact runtimes, static partitioning, Tetris multi-resource packing [17] and critical-path-first ranking [18]—complemented

by ablation, parameter sensitivity, platform sweeps and robustness tests. On the default 256-core cluster, Pass shortens the makespan of IDA batches by 33.9% and reduces the mean waiting time of online campaigns by 89.6% relative to the best baseline, raises core utilization from at most 57.8% to 74.6–79.5%, and needs about 142µs per scheduling decision.

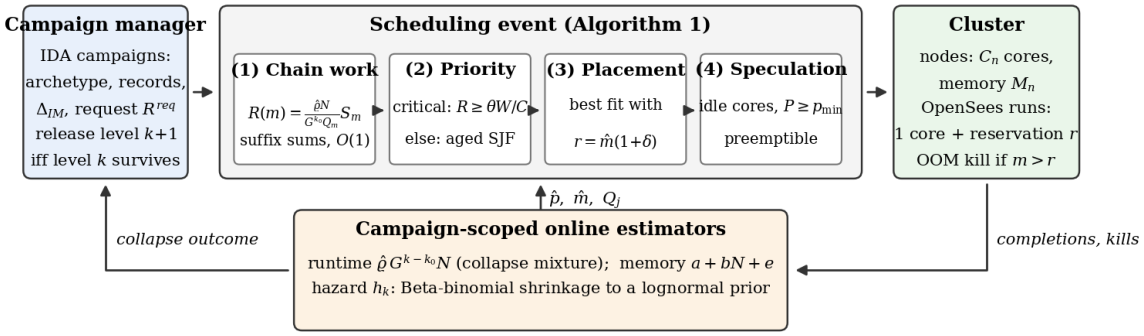


Figure 1. Architecture of Pass. Completed OpenSees runs update campaign-scoped runtime, memory and collapse-hazard models; the scheduler orders ready runs with a bound-guided rule, right-sizes memory reservations and fills idle cores with preemptible speculative runs of future intensity levels.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 characterizes the measured workload. Section 4 formulates the scheduling problem and Section 5 presents Pass. Sections 6 and 7 describe the methodology and results, and Section 8 discusses deployment and concludes the paper.

2. Background and Related Work

2.1. Simulation Campaigns in Earthquake Engineering

IDA [2] subjects a structural model to a set of ground motions, each scaled to increasing intensity measure levels, and records the engineering demand until global dynamic instability; practical guidance on the stepping and hunt-and-fill tracing algorithms is given in [3]. The FEMA P695 methodology [1] applies IDA to archetype models under a far-field set of 22 record pairs to quantify seismic performance factors, and collapse simulation relies on component models with strength and stiffness deterioration [19], [20] and on fiber-based frame elements for reinforced-concrete members [21]. Methods that reduce the number of analyses, such as multiple-stripe analysis with efficient fragility fitting [22] or the imputation of omitted IDA runs from a computed subset, which halves the number of analyses for the FEMA P695 records [23], are complementary to scheduling: they change which analyses are requested, whereas this paper concerns how the requested analyses of many campaigns share a cluster.

OpenSees [4] and OpenSeesPy [5] are the de facto platforms for such studies, and Python tooling increasingly couples analyses with in-memory post-processing [24]. Vamvatsikos [10] distributed single- and multi-record IDA studies over networked processors through master–worker task farming with dynamic task allocation and reported near-linear speedups on dedicated processors. At larger scale, the NHERI SimCenter platform executes building-level analyses for regional risk studies on high-performance computing resources [7], and EQSIM couples regional ground-motion and structural simulations on leadership-class systems [6]. These efforts parallelize one study on dedicated resources; they do not address the multi-campaign, multi-user setting in which the order of runs, the memory reservation of each run and the use of temporarily idle cores determine waiting time, makespan and utilization.

2.2. Batch, Workflow and Multi-Resource Scheduling

FCFS with EASY backfilling [8] remains the standard policy of production resource managers such as SLURM [9]. Users' runtime estimates are inaccurate, and system-generated predictions based on recent history—e.g., the average of a user's last two jobs—improve backfilling [11]. Recent characterizations of production systems show that most jobs use only a fraction of the memory they are allocated [12], and recent evaluation studies show that single averaged objectives can hide important effects, recommending utilization-oriented criteria [25]. List scheduling of directed acyclic graphs (DAGs) ranks tasks by their upward rank, i.e., the length of the longest path to an exit task, as in HEFT [18]; Tetris [17] packs multi-resource demands by an alignment score combined with shortest-remaining-work ordering; and pilot systems such as RADICAL-Pilot execute tens of thousands of heterogeneous tasks inside a single allocation [26]. These approaches treat jobs as opaque or assume that the DAG is known at submission. In IDA the graph is conditional: the existence of a successor depends on the outcome (collapse or not) of its predecessor, so the upward rank is a random variable that must be estimated online.

2.3. Speculative Execution and Resource Sizing

Speculative or backup execution in data-parallel frameworks such as LATE [27] duplicates slow tasks to mitigate stragglers. Pass uses speculation for a different purpose: it pre-executes control-dependent successors whose necessity is uncertain. On the resource side, the job-sizing strategy of Tovar et al. [13] trades failed under-sized attempts against over-allocation, and Sizey [14] and Ponder [15] predict the memory of recurring workflow tasks online and retry under-sized attempts. Pass adopts a retry-based sizing scheme but derives the reservation from a footprint model specific to response-history analyses (model size and output volume).

2.4. Uncertainty-Aware Predictive Scheduling

The closest conceptual precedent of Pass is PRESAGE [16], an uncertainty-aware predictive elastic scheduler for cloud analytics and inference services. PRESAGE establishes a design principle that we adopt throughout this paper: predictive uncertainty should not be reported as a separate, descriptive output, but coupled directly to the scheduling and allocation decision. Container orchestration under cloud uncertainty follows a related idea by adapting resource configurations to observed uncertainty [28]. Pass transfers the probability-to-decision coupling of PRESAGE to structural-simulation campaigns on shared clusters, where three aspects differ: the uncertain quantity is whether a future IDA level will be needed at all, and its runtime and footprint; the decisions are speculative pre-execution, critical-chain priority and memory reservation rather than elastic scaling; and the probabilities are estimated online from the structural outcomes (collapse or survival) of the campaign itself. Table 1 summarizes how Pass relates to these lines of work: it is, to our knowledge, the first scheduler that combines outcome-dependent task chains, structured runtime prediction, memory right-sizing and probability-gated speculation for shared clusters.

Table 1. Positioning of Pass with respect to representative related approaches (✓: supported; ○: partially supported; –: not addressed; Unc.-to-decision: predictive uncertainty drives allocation decisions).

Approach	DAG deps.	Conditional tasks	Prediction	Resource sizing	Unc.-to-decision	Speculation	Multi-user
Parallel IDA task farming [10]	✓	✓	–	–	–	–	–
FCFS/EASY backfilling [8], [9]	–	–	○	–	–	–	✓
History-based runtime prediction [11]	–	–	✓	–	–	–	✓
HEFT list scheduling [18]	✓	–	–	–	–	–	–
Tetris multi-resource packing [17]	–	–	○	–	–	–	✓
Straggler speculation (LATE) [27]	–	–	○	–	–	✓	✓
Workflow memory sizing [13-15]	○	–	✓	✓	○	–	✓
Uncertainty-aware container orchestration [28]	–	–	○	✓	✓	–	✓
PRESAGE predictive elastic scheduling [16]	–	–	✓	✓	✓	–	✓
Pass (this work)	✓	✓	✓	✓	✓	✓	✓

3. Measurement-Based Workload Characterization

3.1. Measurement Setup

Archetypes. We built four parametric 2D archetypes in OpenSeesPy 3.8.0, the latest release at the time of the measurements. SMF4, SMF8 and SMF12 are four-, eight- and twelve-story, three-bay steel special moment frames (bay width 6.1 m, story heights 4.6/4.0 m) designed with an equivalent-lateral-force procedure ($S_{DS} = 1.0$ g, $S_{D1} = 0.6$ g, $R = 8$): beam plastic moments follow portal-method demands with a design overstrength of 2.0, column strengths satisfy a strong-column/weak-beam ratio of 1.5, and member stiffnesses are scaled to the target first-mode period. Plasticity is concentrated in modified Ibarra–Medina–Krawinkler springs with cyclic deterioration [19], [20] at beam ends and column bases, using the common $n = 10$ stiffness modification of the elastic members, with stiffness-proportional damping assigned to the elastic members only. RCF4 is a four-story, three-bay reinforced-concrete frame with force-based fiber beam-columns (confined and unconfined Concrete02, Steel02 reinforcement) in the spirit of [21]. All models include a leaning column with corotational trusses for P-Δ effects and 2.5% Rayleigh damping. The fundamental periods are 1.34, 2.09, 2.63 and 0.93 s for SMF4, SMF8, SMF12 and RCF4.

Records and IDA protocol. We use the 22 FEMA P695 far-field records [1] (first horizontal component, PEER NGA record sequence numbers; durations 20–100 s, time steps 0.0025–0.02 s), obtained from an open-source distribution of the record set. Each record is scaled to $S_a(T_1, 5\%) = k\,\Delta_{\mathrm{IM}}$ with $\Delta_{\mathrm{IM}} = 0.15$, 0.09, 0.08 and 0.28 g for SMF4, SMF8, SMF12 and RCF4, chosen from a pilot study so that the median collapse occurs near the eighth step. Analyses use Newmark average-acceleration integration with $\Delta t = 0.01$ s over the record plus 2 s of free vibration. Collapse is declared when the peak interstory drift exceeds 10% or when the fallback sequence (Newton, Newton with line search, Krylov–Newton, modified Newton, and step subdivision down to $\Delta t/50$) fails. Stepping continues until the first collapse; two further levels are executed and recorded so that speculative execution can be replayed faithfully.

Output profiles. Two common post-processing styles are represented. Collapse-only (C) runs record floor displacements and compute drift demands after the analysis. Full-field (F) runs additionally record all nodal displacements, floor accelerations, element forces and hinge deformations and load them into memory for envelope and cumulative-rotation post-processing, as Python-based workflows do [24]. The four campaign types are SMF4-F, SMF8-F, SMF12-C and RCF4-C.

Instrumentation. Every analysis runs as an isolated operating-system process; its CPU time (user+system) and peak resident set size are obtained from wait4(). The resident set of selected runs was additionally sampled every 20 ms. In total we measured 810 analyses (636 needed runs) consuming 1.00 CPU-hours, plus 68 repeated executions. Table 2 summarizes the trace.

Table 2. Measured OpenSees IDA trace (22 FEMA P695 far-field records per archetype). Runs include two levels beyond the first collapse; needed runs in parentheses.

Type	T1 (s)	ΔIM (g)	runs (needed)	K mean [range]	med. Sa, col (g)	runtime med./p95 (s)	RSS med./max (MB)	request (MB)
RCF4-C	0.93	0.28	146 (102)	4.6 [2–10]	1.12	5.33 / 10.85	88 / 89	192
SMF12-C	2.63	0.08	210 (166)	7.5 [4–13]	0.64	5.86 / 11.64	88 / 91	192
SMF4-F	1.34	0.15	258 (216)	9.8 [4–20]	1.50	2.25 / 4.02	110 / 132	256
SMF8-F	2.09	0.09	196 (152)	6.9 [3–12]	0.63	4.53 / 8.56	123 / 174	320

3.2. Observations

Fig. 2 and Fig. 3 illustrate the main properties of the workload.

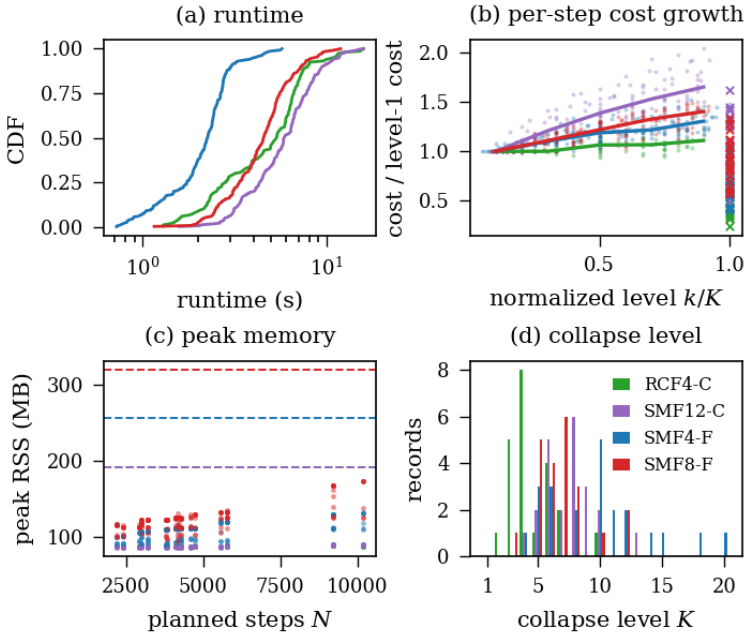


Figure 2. Measured OpenSees trace; colors identify the campaign types listed in (d). (a) Runtime distribution per campaign type. (b) Per-step cost relative to level 1 of the same record versus normalized level k/K (dots, median lines); crosses show the full runtime of collapse runs relative to level 1. (c) Peak memory versus planned steps; dashed lines are the static requests. (d) Distribution of collapse levels.

O1: Runtime is heterogeneous but structured. Task runtimes span 0.7–15.8 s (coefficient of variation 0.59 over all runs). Within a campaign type, most of the variation is explained by the planned number of steps, i.e., by the record duration known before execution (Fig. 2a,c).

O2: Intensity amplifies cost; collapse truncates it. The per-step cost grows with intensity as yielding increases Newton iterations and triggers fallback algorithms; at the last non-collapse level it reaches a median of 1.25× the level-1 cost (Fig. 2b). Collapse runs stop when the drift limit is reached and cost a median of 0.56× the preceding level. A predictor must therefore combine record length, a per-campaign cost scale, intensity growth and the probability of collapse—exactly the structure of (2).

O3: Chain lengths are uncertain, and chain parallelism decays. Collapse levels range from 2 to 20 (Fig. 2d), with mean values of 4.6, 7.5, 9.8 and 6.9 (RCF4-C, SMF12-C, SMF4-F and SMF8-F) for

the four types. Executed with unlimited cores, the 88 chains of the trace finish one by one: half of them have terminated at 30% of the longest chain, and the number of active chains stays below 64 for most of the campaign (Fig. 3). The longest chain is 2.1× longer than the load bound W/C of a 64-core cluster, so the makespan of an IDA batch is bounded by chains rather than by cores, and idle cores are inevitable without speculation.

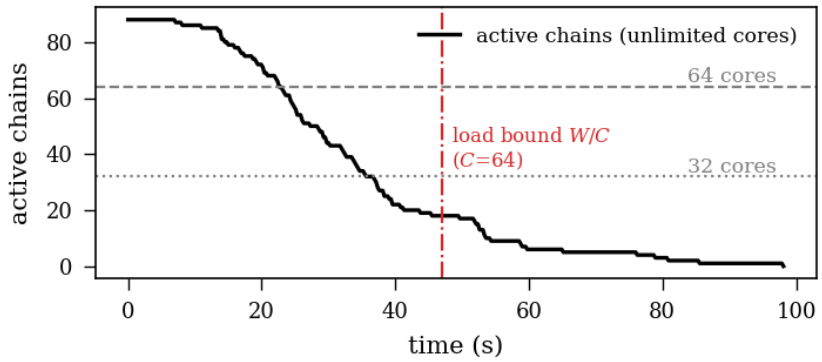


Figure 3. Active chains over time when all 88 measured chains run with unlimited cores and no speculation. Chain parallelism drops below typical core counts long before the longest chain completes; the load bound W/C for 64 cores is far below the longest chain.

O4: Memory is predictable and over-requested. The coefficient of variation of the peak resident set is 0.005/0.010 within collapse-only campaigns and 0.081/0.117 within full-field campaigns, where it grows linearly with the planned steps (Fig. 2c). Static requests that cover the largest footprint with a 50% margin exceed the actual footprint by a median factor of 2.28. The resident set of full-field runs peaks during post-processing at 93% of the runtime, whereas collapse-only runs reach their peak within the first 4%, which determines how much work an OOM kill wastes.

O5: Generic predictors fail on this structure. Table 3 replays each campaign type 20 times in random asynchronous order and predicts every run online before it is revealed. The last-two-runs predictor of [11] and the campaign mean are essentially uncorrelated with the actual runtime (Spearman $\rho \approx 0$), so SJF ordering based on them is close to random. Scaling by planned steps restores the ranking, and the full model (2) further reduces the median error to 5.9% (versus 26.0% for the last-two predictor). The online hazard (3) also improves the Brier score of per-level collapse predictions from 0.121 (prior only) to 0.113.

Table 3. Online runtime prediction on the trace (mean of 20 asynchronous replays per campaign type): median and 90th-percentile absolute percentage error and Spearman rank correlation with the actual runtime.

Predictor	MdAPE (%)	P90 APE (%)	ρ
Last two runs [11]	26.0	82.0	0.05
Campaign mean	19.2	75.8	0.00
Steps-scaled median	17.9	60.9	0.71
Pass model (2)	5.9	44.6	0.77

4. Problem Formulation

Platform. A cluster consists of nodes $n \in \mathcal{N}$, each with C_n cores, memory capacity M_n and relative speed s_n; $C = \sum_n C_n$ denotes the total number of cores. Every structural analysis is a single-core process, which matches the common practice of running independent OpenSees interpreters per analysis [10].

Campaigns and conditional chains. Campaign c arrives at time a_c and comprises a set $\mathcal{G}_c$ of ground motions. Each record $g \in \mathcal{G}_c$ induces a chain of tasks $\tau_{c,g,1}, \tau_{c,g,2}, \ldots$ at intensity levels $\mathrm{IM}_k = k\,\Delta_{\mathrm{IM}}$, $k = 1, \ldots, k_{\max}$. Task $\tau_{c,g,k}$ has runtime $p_{c,g,k}$ on a unit-speed core, peak memory $m_{c,g,k}$ and

collapse outcome $y_{c,g,k} \in \{0,1\}$. Let $K_{c,g} = \min\{k: y_{c,g,k} = 1\}$ (or $k_{\max}$ if the record never collapses). Task $\tau_{c,g,k}$ is needed if and only if $k \leq K_{c,g}$, i.e., all lower levels survived. The quantities p, m, y and hence K are unknown before execution; only the planned number of integration steps $N_{c,g}$ (record duration plus free vibration divided by the analysis step) and the model are known.

Execution model. A task is started on a node with one free core and a memory reservation $r \leq$ free memory. If $r < m$, the task is killed when its resident set crosses r—at fraction κ of its runtime—and must be retried, as enforced by cgroup-based memory limits of resource managers. A task may be started speculatively before it is known to be needed; such a run is wasted if a lower level of its chain collapses and may be preempted (killed) at any time.

Metrics. Let f_c be the completion time of the last needed task of campaign c and $\rho_{c,g,k}$ the time at which a needed task becomes ready (arrival for $k = 1$, completion of level $k - 1$ otherwise). We evaluate

- makespan $C_{\max} = \max_c f_c - \min_c a_c$;
- mean waiting time $\bar{W} = \text{mean}(\sigma_{c,g,k} - \rho_{c,g,k})$ over needed tasks, where σ is the start time of the successful run (zero if the run was already executing speculatively when it became needed), together with its 95th percentile;
- core utilization $U = \sum(_{\text{busy core-seconds}})/(C\, C_{\max})$ and useful utilization U*, which counts only runs whose results are consumed;
- memory efficiency $E_M = \int_{\text{used memory}} dt / \int_{\text{reserved memory}} dt$;
- on-time completion rates: the fraction of needed tasks completed before the deadline $d_c = a_c + \lambda\, \text{CP}_c$ of their campaign and the fraction of campaigns completed before it, where CP_c is the critical path of the campaign (the largest total runtime of a needed chain), i.e., the earliest completion that is possible without speculation; we use $\lambda = 1$ for tasks and $\lambda = 1.25$ for campaigns;
- scheduling overhead, the wall-clock latency of each scheduling decision.

Lower bounds. For any schedule without speculation, $C_{\max} \geq \max\{W/C,\ \max_{c,g} \sum_{k \leq K_{c,g}} p_{c,g,k}\}$ for a batch of campaigns released at time zero, where W is the total needed work [29]. The first term is reached only if the cluster stays busy; the second is the longest chain. Generic list schedulers approximately attain the first term while many chains are active but are bounded by the second term in the drain phase. Speculation is the only mechanism that can beat the second term, because it overlaps levels of the same chain.

5. The Pass Scheduler

5.1. Overview

Fig. 1 shows the structure of Pass. A campaign manager expands each submitted IDA campaign into chains and releases level $k + 1$ of a chain when level k completes without collapse. Completed runs feed three campaign-scoped estimators: a runtime model, a memory model and a collapse-hazard model. Every scheduling event (arrival, completion, kill) triggers four steps: (i) refresh the survival-weighted remaining work of the chains whose campaign changed; (ii) order the ready queue with the bound-guided priority rule; (iii) place tasks with right-sized memory reservations (best fit), preempting speculative runs if confirmed work is blocked; (iv) fill the cores that remain idle with probability-gated speculative runs. All estimators are scoped to the campaign being estimated, so that no information about other campaigns' records leaks into a prediction; the only cross-campaign quantity is a cold-start median of the per-step cost.

5.2. Intensity-Conditioned Runtime and Memory Model

For a run that does not collapse, the cost is the product of the planned number of steps, the per-step cost of the model and an intensity-dependent amplification caused by additional Newton iterations and step subdivision. We write

$$\hat{p}^{\mathrm{nc}}_{c,g,k} = \hat{\varrho}_{c,g}\; G_c^{\,k-k_0}\; N_{c,g} \tag{1}$$

Where $\hat{\varrho}_{c,g}$ is the per-step cost observed at the last completed non-collapse level k_0 of the same chain. If the chain has no completed level, $\hat{\varrho}$ is the campaign median of the level-normalized per-step cost $\varrho_{c,g,k}/G_c^{k-1}$, and before the first completion of a campaign it is the median over all completed runs in the system. The growth factor G_c is the median ratio of per-step costs of consecutive non-collapse levels of the same chain (prior value 1.05). A collapse run terminates early; its cost is a fraction f_c of the non-collapse prediction, with f_c the median observed ratio (prior 0.5). Given the hazard $h_{c,k}$ of collapsing at level k (Section 5.3), the expected runtime is

$$\hat{p}_{c,g,k} = \hat{p}^{\mathrm{nc}}_{c,g,k}\,\Big(1 - h_{c,k}(1 - f_c)\Big) \tag{2}$$

The memory model is a robust linear fit of the peak resident set against the planned steps, $\hat{m}_{c,g} = a_c + b_c N_{c,g} + e_c$, where e_c is the largest positive residual observed in the campaign. The intercept captures the interpreter and model state, the slope captures recorded response histories that are post-processed in memory.

5.3. Online Collapse Hazard and Survival-Weighted Chain Work

Let $n_{c,k}$ be the number of chains of campaign c whose level k has been confirmed (needed and completed) and $d_{c,k}$ the number of those that collapsed at level k. The discrete hazard is estimated by Beta–binomial shrinkage toward a prior derived from a lognormal fragility in level space with median μ_0 and dispersion β_0 [22]:

$$h_{c,k} = \frac{d_{c,k} + n_0\, h_k^0}{n_{c,k} + n_0} \tag{3}$$

Where $h_k^0 = (F_k - F_{k-1})/(1 - F_{k-1})$ and $F_k = \Phi(\ln(k/\mu_0)/\beta_0)$ is the prior fragility evaluated at level k. The prior median μ_0 encodes the number of IDA steps that the analyst intends to trace to collapse when choosing ΔIM; the pseudo-count n_0 controls how quickly observed collapses dominate. Counting only confirmed levels keeps the estimator a Kaplan–Meier-type product-limit estimator [30] under the right censoring induced by chains that are still running.

Let $Q_{c,j} = \prod_{i<j}\big(1 - h_{c,i}\big)$ be the probability of reaching level j. For a chain whose lowest unconfirmed level is m, the probability that level $j \geq m$ is needed is $Q_{c,j}/Q_{c,m}$, and the survival-weighted expected remaining work is

$$R_{c,g}(m) = \sum_{j=m}^{k_{\max}} \frac{Q_{c,j}}{Q_{c,m}}\,\hat{p}_{c,g,j} = \frac{\hat{\varrho}_{c,g} N_{c,g}}{G_c^{k_0}\, Q_{c,m}}\; S_{c,m} \tag{4}$$

With $S_{c,m} = \sum_{j\geq m} G_c^{\,j}\, A_{c,j} Q_{c,j}$ and $A_{c,j} = 1 - h_{c,j}(1 - f_c)$. Because $S_{c,m}$ is a suffix sum that depends only on campaign-level quantities, it is precomputed once per estimator update in $O(k_{\max})$, after which $R_{c,g}(m)$ of every chain is obtained in $O(1)$. Levels whose speculative results are already available are subtracted from (4).

5.4. Bound-Guided Priority

Let $\mathcal{A}$ be the set of active chains and $W = \sum_{(c,g)\in\mathcal{A}} R_{c,g}$ the expected outstanding work. Following the lower bound of Section 4, a chain whose remaining work exceeds the per-core share of the outstanding work cannot finish by the load-balanced completion time, however the other tasks are ordered. Pass classifies the ready task of a chain as critical if

$$R_{c,g} \geq \theta\, W/C \tag{5}$$

And orders the ready queue lexicographically: critical tasks first, by decreasing $R_{c,g}$ (longest expected remaining chain first); then non-critical tasks by the aged expected runtime $\hat{p}_{c,g,k} - \omega\left(t - \rho_{c,g,k}\right)$, which approximates shortest-job-first while bounding starvation through the aging rate ω. While many chains are active, W/C is large and almost no chain is critical, so Pass behaves as an estimate-driven SJF that minimizes waiting time; as campaigns drain, W/C shrinks, the longest chains become critical and are prioritized, which shortens the tail. The parameter θ moves the switching point.

5.5. Probability-Gated, Preemptible Speculation

When the ready queue is empty or blocked and cores are idle, Pass launches future levels of active chains. For each chain whose lowest unconfirmed level m is running, the candidate is the lowest level $j \in (m, m+D]$ that is neither running nor completed; it is eligible if

$$P_{c,g,j} = Q_{c,j}/Q_{c,m} \geq p_{\min} \tag{6}$$

And candidates are launched in decreasing order of $P_{c,g,j}\, R_{c,g}$, i.e., probable levels of long chains first. Gate (6) is the point at which Pass applies the probability-to-decision coupling of PRESAGE [16]: the online estimate of task necessity is not merely reported, it decides whether an idle core executes a future level. A speculative run that finishes before its predecessor keeps its result; the result is committed as soon as all lower levels are confirmed. If a lower level collapses, the running speculative levels of the chain are cancelled and held results are discarded. Whenever a confirmed ready task cannot be placed, Pass preempts the speculative run with the lowest P (latest start first) on a node where the freed reservation suffices.

Proposition 1 (Non-intrusiveness). With immediate preemption, the start of a confirmed task is never delayed by speculative runs: at every scheduling event, each confirmed ready task that would be placed in the absence of speculative runs is placed.

Proof sketch. Speculative runs are only started on cores that remain idle after all placeable confirmed tasks have been started. When a confirmed task cannot be placed, the preemption step releases the core and reservation of a speculative run on a node that can host it; the freed node state is identical to the state without that speculative run, hence the placement decision is the same.

Proposition 2 (Chain acceleration). Consider a chain whose levels have equal runtime τ and whose first collapse occurs at level K, executed with unlimited idle cores and $P \geq p_{\min}$ for all levels. Without speculation the chain completes at $K\tau$. With speculation depth D it completes at $\lceil K/(D+1) \rceil \tau$ and wastes at most $D\tau$ core-seconds.

Proof sketch. At the start of each window of length τ, levels $m, \ldots, m+D$ run concurrently; at the end of the window they are all confirmed if none collapses, so the chain advances by $D+1$ levels per window. In the window containing K, the at most D levels above K are discarded.

Proposition 2 explains why speculation addresses the drain phase directly: when chains, not cores, are the scarce resource, each idle core converts into chain progress with probability P, and the gate (6) bounds the expected waste per speculative run by $(1 - p_{\min})$ of its runtime.

5.6. Memory Right-Sizing with OOM Feedback

For a campaign whose memory model has at least three observations, the reservation is

$$r_{c,g} = \min\{ R_c^{\mathrm{req}},\ \hat{m}_{c,g}\,(1 + \delta_c)\} \tag{7}$$

Where R_c^{req} is the user request; before three observations the request is used. After an OOM kill the task is retried with the user request, and the campaign margin is increased, $\delta_c \leftarrow \min(1, \delta_c + 0.1)$; every successful run decays it, $\delta_c \leftarrow \max(0.05, 0.995\,\delta_c)$. Following the cost argument of job sizing [13], the expected reservation cost of a run with runtime p is $r\,p + \Pr[m > r]\,(\kappa\,r\,p + R^{\mathrm{req}}p)$; since footprints of response-history analyses are tightly determined by the model and the output volume (Section 3), $\Pr[m > r]$ is small and the first term dominates.

5.7. Algorithm and Complexity

Algorithm 1 summarizes one scheduling event. Estimator updates cost $O(k_{\max})$ per completion and a refit of the memory model every few observations. Refreshing remaining work costs $O(|\mathcal{G}_c|)$ for each campaign touched by the event, ordering the ready queue $O(|\mathcal{R}|\log|\mathcal{R}|)$, placement $O(|\mathcal{N}|)$ per started task and speculation $O(|\mathcal{K}| + F\log|\mathcal{K}|)$ for F idle cores and $|\mathcal{K}|$ cached candidates. Section 7 reports measured decision latencies.

Algorithm 1. Pass scheduling event at time t

6. Experimental Methodology

Trace-driven simulation. We evaluate all policies with a discrete-event cluster simulator that replays the measured OpenSees runs of Section 3: the runtime, peak memory and collapse outcome of every task are those of the corresponding real analysis, multiplied by a run-to-run variability factor drawn from 68 repeated executions of randomly selected runs (median ratio 1.011, 5–95% range 0.95–1.09). OOM kills occur at the measured fraction of runtime at which the resident set reaches its peak (0.93 for full-field and 0.04 for collapse-only runs). The simulator, the schedulers and the measurement harness are written in Python and released with the trace.

Platform. The default cluster has 8 nodes with 32 cores each (256 cores). Because the absolute footprints of the 2D archetype analyses are small, node memory is expressed relative to the users' requests through the memory-pressure ratio $\psi = \bar{R}^{\mathrm{req}}/M_{\mathrm{core}}$, where $\bar{R}^{\mathrm{req}}$ is the mean static request and M_{core} the memory per core; $\psi \le 1$ means that static requests never constrain packing and $\psi > 1$ represents memory-lean nodes, as encountered when large models run on high-core-count nodes. The default is $\psi = 1.25$ ($M_{\mathrm{core}} = 192$ MB); Section 7.4 varies ψ from 0.5 to 2.0. Static requests follow common practice: each campaign requests its largest footprint plus a 50% margin, rounded up to 64 MB; walltime requests for EASY are twice the largest runtime of the campaign type, rounded up to one minute.

Workloads. A campaign is an IDA of one archetype (chosen uniformly) under a random subset of 14–22 of the 22 records. Batch workloads release 20 campaigns at time zero, representing a multi-archetype study such as a FEMA P695 evaluation. Online workloads submit 60 campaigns from 4 user groups with Poisson arrivals whose rate yields an offered load of $\rho = 0.85$ of the cluster capacity (computed from needed work only). Each configuration is repeated with 10 random seeds (5 for sweeps), which change campaign composition, record subsets, arrival times and runtime variability; we report means and standard deviations.

Baselines. All baselines reserve the static memory request. FIFO: strict FCFS with first-fit placement. EASY: FCFS with EASY backfilling [8] using walltime requests. SJF: shortest-job-first with the history predictor of [11] (mean of the last two runs of the same campaign). SJF-Oracle: shortest-job-first with exact runtimes, a reference that no deployable scheduler can match. STATIC: static resource allocation; the cluster is partitioned into four equal node groups, one per user group, scheduled FCFS. Tetris: multi-resource packing by the alignment score combined with shortest-remaining-work ordering [17]. CPF: critical-path-first list scheduling with a collapse-agnostic upward rank $(k_{\max} - k + 1)\,\hat{p}$, the HEFT ranking [18] applied to the full intensity ladder.

Pass configuration. Unless stated otherwise $\theta = 1$, $\omega = 0.02$, $p_{\min} = 0.5$, $D = 2$, $\mu_0 = 8$, $\beta_0 = 0.5$, $n_0 = 3$, $\delta_0 = 0.1$, and $k_{\max} = 20$. These values follow from the design rationale ($\theta = 1$ is the lower-bound crossover, μ_0 is the number of tracing steps targeted when ΔIM was chosen) and were not tuned on the evaluation seeds; Section 7.3 varies each of them.

Uncertainty-to-scheduling evaluation. We implement the probability-to-decision coupling established as a useful design principle by the PRESAGE framework [16]: rather than treating uncertainty as a separate descriptive output, Pass converts the online, survival-weighted probability that an IDA level will be required into a concrete speculative-execution decision through gate (6). The measured-trace evaluation tests this implementation through the online-hazard-update experiment (Table 6) and the speculation-gate sweep (Fig. 6). In both tests, the scheduling action is driven by estimated task necessity, which gives a direct experimental assessment of uncertainty-informed resource allocation for structural-simulation chains.

Metrics. Because a single averaged objective can hide important effects in batch scheduling [25], we report the complementary metrics of Section 4: makespan, mean and 95th-percentile waiting time, core utilization U and useful utilization U*, memory efficiency E_M, task and campaign on-time completion rates ($\lambda = 1$ and 1.25), mean campaign turnaround, wasted core time and the per-decision scheduling latency measured with a monotonic wall clock on a single 2.1 GHz core.

7. Results

7.1. Overall Comparison

Table 4 and Fig. 4 summarize the default configuration.

Table 4. Main results on the default 256-core cluster ($\psi = 1.25$), mean over 10 seeds (± standard deviation). U: core utilization; U*: useful utilization; E_M: memory efficiency; task on-time: needed tasks finished before the campaign arrival time plus its critical path; camp. on-time: campaigns finished within $1.25\times$ their critical path; overhead: mean latency per scheduling decision. Best deployable value in bold.

Policy	Makespan (s)	Wait (s)	P95 wait (s)	U (%)	U* (%)	E_M (%)	Task on-time (%)	Camp. on-time (%)	Turnaround (s)	Overhead (µs)
Batch: 20 campaigns released at $t = 0$										
FIFO	111±2	2.1±0.3	4	44.2	44.2	43.8	93.9	66.5	87	14
EASY	110±2	2.0±0.3	4	44.6	44.6	43.8	94.1	66.5	86	79
SJF	113±4	1.5±0.2	9	43.4	43.4	43.8	97.1	80.0	82	58
SJF-Oracle	132±4	1.5±0.2	11	37.3	37.3	43.8	95.0	34.0	92	47
STATIC	131±9	3.2±0.7	8	37.7	37.7	43.8	84.6	44.5	97	41
Tetris	131±6	1.7±0.2	19	37.7	37.7	43.8	95.1	64.5	83	82
CPF	104±1	1.8±0.3	7	47.2	47.2	43.8	86.7	72.0	85	64
Pass	69±3	0.9±0.1	6	74.6	71.5	81.4	99.7	100.0	58	142
Online: 60 campaigns, Poisson arrivals, $\rho = 0.85$										
FIFO	261±16	2.0±1.4	5	56.2	56.2	43.7	91.8	51.0	92	14
EASY	261±17	2.0±1.4	5	56.3	56.3	43.7	92.1	52.7	91	100
SJF	259±18	1.6±1.0	5	56.6	56.6	43.7	94.5	76.0	88	87
SJF-Oracle	258±19	1.2±0.8	4	56.9	56.9	43.7	95.1	57.5	95	44
STATIC	295±15	4.2±2.0	13	49.8	49.8	43.7	81.5	34.2	114	73
Tetris	263±16	1.1±0.8	6	55.9	55.9	43.7	95.2	87.2	78	98
CPF	255±20	2.0±1.3	7	57.8	57.8	43.7	87.2	70.0	92	99
Pass	199±18	0.1±0.2	0	79.5	74.1	68.4	99.7	97.5	48	110

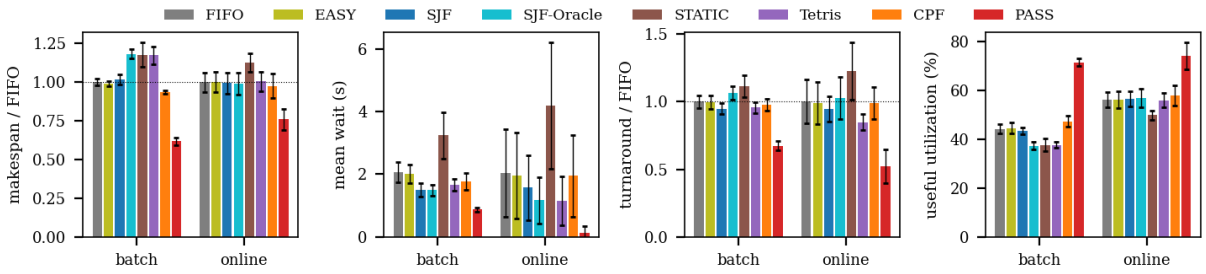


Figure 4. Default configuration (256 cores, $\psi = 1.25$, 10 seeds, error bars: one standard deviation). Makespan and campaign turnaround are normalized to FIFO; the mean waiting time is shown in seconds.

Batch campaigns. All policies without speculation finish an IDA batch between 104 and 132 s, close to the duration of the longest needed chain: once the short chains have collapsed, the remaining chains cannot use the idle cores (Fig. 5). Reordering does not remove this limit and instead exposes the tension between waiting time and makespan. SJF and SJF-Oracle reduce the mean waiting time to 1.5–1.5 s (FIFO: 2.1 s) but lengthen the makespan to 113–132 s (FIFO: 111 s) because they postpone the costly upper levels of long chains. CPF, which favors long chains, finishes in 104 s but cannot overlap the levels of a chain, and STATIC suffers from fragmentation, with idle partitions next to overloaded ones (131 s, 3.2 s mean wait). Pass completes the batch in 69 s, i.e., 33.9% faster than the best baseline (CPF) and 38.1% faster than FIFO, and simultaneously reduces the mean waiting time by 41.6% relative to the best baseline (SJF). Its core utilization rises from at most 47.2% to 74.6% (71.5% useful), and the mean campaign turnaround drops by 29.0%. Of the speculative runs, 64% turned into needed results, and speculation, preemption and OOM retries together consumed only 4.3% of the busy core time. Memory right-sizing raises memory efficiency from 43.8% to 81.4% at the cost

of 0.8 OOM kills per batch of 2,603 needed tasks. FCFS-based policies keep the lowest 95th-percentile waiting time in batch mode (4–4 s); thanks to aging, the value of Pass (6 s) remains below that of every reordering baseline.

Online campaigns. With continuous arrivals, static requests limit the number of co-located analyses, and the baselines operate close to saturation. Pass serves the same offered load with right-sized reservations: the mean waiting time drops by 89.6% relative to the best baseline (Tetris) and the 95th percentile from 4–13 s to 0.4 s. Because speculative runs accelerate every active chain, the mean campaign turnaround falls by 38.5% (versus Tetris) and by 47.8% versus FIFO, and utilization rises from at most 57.8% to 79.5%. The deadline-based completion rates follow the same pattern: Pass finishes 99.7% of the needed tasks before their campaign's critical-path deadline (best baseline: 95.2%) and 97.5% of the campaigns within $1.25\times$ their critical path (best baseline: Tetris, 87.2%; FIFO: 51.0%). In batch mode, 100.0% of the campaigns of Pass meet the $1.25\times$ deadline, compared with at most 80.0% for the baselines.

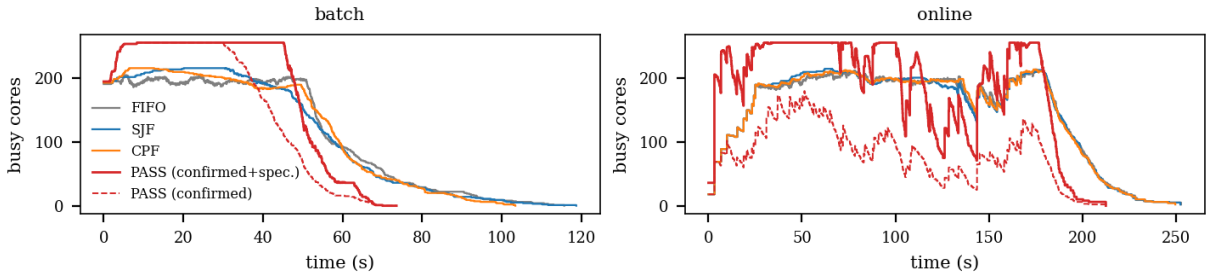


Figure 5. Busy cores over time for one seed. Without speculation, the number of busy cores of every baseline collapses after the short chains finish; Pass keeps the cores busy with speculative levels (difference between solid and dashed red lines) and finishes the batch earlier.

7.2. Ablation Study

Table 5. Ablation of Pass components (mean over 10 seeds; relative change to the full scheduler in parentheses).

	Batch			Online				
2-4(lr)5-9 Variant	Makespan (s)	Mean wait (s)	Waste (%)	Makespan (s)	Mean wait (s)	Turnaround (s)	E_M (%)	OOM kills
Pass (full)	69	0.87	4.3	199	0.12	48	68.4	7.8
w/o speculation	101(+47%)	0.86(-1%)	0.1	252(+27%)	0.41(+243%)	78(+62%)	78.0	6.7
w/o critical-chain	74(+7%)	0.80(-8%)	4.7	200(+1%)	0.09(-22%)	48(+0%)	68.5	10.2
w/o aged-SJF	71(+3%)	1.14(+31%)	3.6	199(+0%)	0.13(+12%)	48(-1%)	68.5	6.9
w/o runtime model	70(+1%)	0.94(+8%)	4.5	200(+1%)	0.14(+13%)	48(-1%)	70.0	14.2
w/o hazard update	64(-7%)	0.87(-0%)	4.3	195(-2%)	0.12(-2%)	47(-3%)	68.7	8.3
w/o memory right-sizing	77(+13%)	1.49(+71%)	3.8	214(+8%)	1.31(+988%)	74(+54%)	43.7	0.0
w/o OOM feedback	69(-0%)	0.87(+0%)	4.2	199(+0%)	0.12(-1%)	48(+0%)	67.7	3.0

Table 5 removes one component at a time. Speculation and memory right-sizing are the dominant components. Without speculation, the batch makespan grows by 47.2% and the online turnaround by 62.4%, because the drain phase is no longer filled; the online mean waiting time also rises from 0.12 to 0.41 s, since a level that becomes needed is no longer found already running. Without memory right-sizing, static requests cap the number of co-located runs: the batch mean waiting time grows by 70.5%, the online mean waiting time rises from 0.12 to 1.31 s, and speculation itself is throttled because fewer runs fit into memory. The two priority components act on the trade-off between waiting time and makespan: aged SJF ordering lowers the batch mean waiting time from 1.14 to 0.87 s, whereas critical-chain priority shortens the batch makespan by 7.0% and lowers the speculative waste

from 4.7% to 4.3% of the busy time, at the price of a 9% longer mean wait, because long chains are served first. The intensity-conditioned runtime model improves both makespan (1.5%) and mean wait (8.0%) over the history-based estimate. The online hazard update is not required for good performance: it sharpens the necessity estimates (Brier score 0.113 instead of 0.121 for the prior alone, Section 3) but makes speculation more conservative. With the static lognormal prior ($\mu_0 = 8$), the scheduler launches 6% more speculative runs, raises core utilization from 74.6% to 80.6% and finishes the batch 7.3% earlier, because the prior is optimistic for the RCF4-C chains, which collapse after 4.6 levels on average, and speculation on otherwise idle cores is inexpensive. Practitioners can therefore use either the static prior or the online estimate; the speculation gate $p_{\min}$ controls the same aggressiveness directly (Section 7.3). The adaptive OOM margin changes the results by at most 1%. Table 6 extends this comparison to other prior medians.

7.3. Sensitivity to Pass Parameters

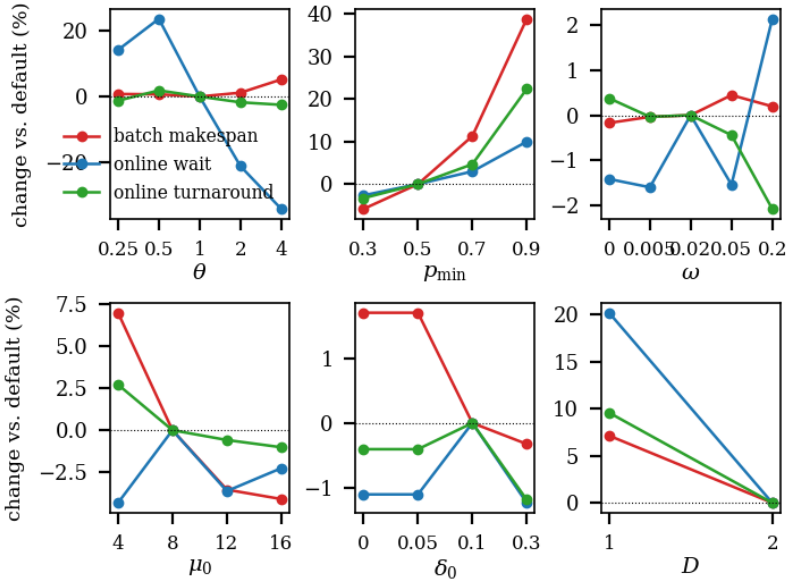


Figure 6. Sensitivity of Pass to its parameters (5 seeds): relative change of batch makespan, online mean waiting time and online turnaround with respect to the default value of each parameter.

Fig. 6 varies one parameter at a time around the defaults. Criticality threshold θ. Smaller thresholds mark more chains as critical: $\theta \leq 1$ yields the shortest batch makespan (69.7–70.2 s), whereas $\theta \geq 2$ lowers the batch mean waiting time from 0.94 to 0.86–0.89 s at an up to 5% longer makespan. The rule thus exposes the waiting-time/makespan trade-off through a single, interpretable knob, and $\theta = 1$, the crossover of the two lower-bound terms, is a balanced default. Thresholds below 1 raise the batch mean waiting time to 1.17–1.21 s without shortening the makespan, which supports $\theta = 1$ as the default. Speculation gate $p_{\min}$. The gate is the decision threshold of the probability-to-decision coupling adopted from PRESAGE [16], and it trades waste for chain acceleration. Raising $p_{\min}$ from 0.5 to 0.9 cuts the speculative waste in batch mode from 4.2% to 0.8% of the busy time but lengthens the makespan from 69.7 to 96.7 s; lowering the gate to $p_{\min} = 0.3$ shortens the makespan to 65.6 s at a waste of 4.5%, so the gate is the single knob that sets how aggressively idle cores are used. Speculation depth D. Depth 2 shortens the batch makespan by 7% and the online turnaround by 9% relative to depth 1, consistent with Proposition 2, at a waste of 4.2% instead of 2.2% of the busy time. Aging rate ω. Results change by at most 3% between $\omega = 0$ and 0.2. Initial memory margin δ_0. Without a margin, the online runs suffer 21.8 OOM kills per experiment; $\delta_0 = 0.1$ reduces them to 6.8 at a memory efficiency of 69%, while $\delta_0 = 0.3$ eliminates them but lowers the efficiency to 65%. Prior median μ_0. A pessimistic prior ($\mu_0 = 4$) makes speculation more conservative (waste 2.9% instead of 4.2%) and lengthens the batch makespan by 7%, whereas more optimistic priors ($\mu_0 = 12$–16) shorten it by up to 4%, consistent with the ablation: when idle cores are available, a prior that keeps speculation aggressive pays off. Table 6 repeats the batch and online experiments with and

without the online hazard update for prior medians between 3 and 16: the batch makespan stays between 63 and 74 s and the two variants differ in batch makespan by at most 7%, so Pass remains effective without an accurate estimate of the collapse capacity. The static prior is at least as fast for every median, which again indicates that keeping speculation aggressive matters more than calibrating the necessity probabilities when idle cores are available.

Table 6. Effect of the online hazard update under (mis)specified prior medians μ_0 (5 seeds). Each cell: with / without hazard update.

	Batch				Online			
2-5(lr)6-9 μ0	makespan (s)		waste (%)		turnaround (s)		waste (%)	
3	73.8	71.4	2.6	1.9	41.8	41.0	5.5	4.7
5	73.1	69.9	3.1	3.2	40.1	39.8	6.4	6.4
8	68.4	63.8	4.3	4.3	39.1	37.5	7.6	7.9
12	66.4	63.1	4.4	4.4	38.4	37.8	7.8	8.0
16	66.9	63.1	4.4	4.4	38.7	37.6	8.0	8.1

7.4. Platform and Load Sweeps

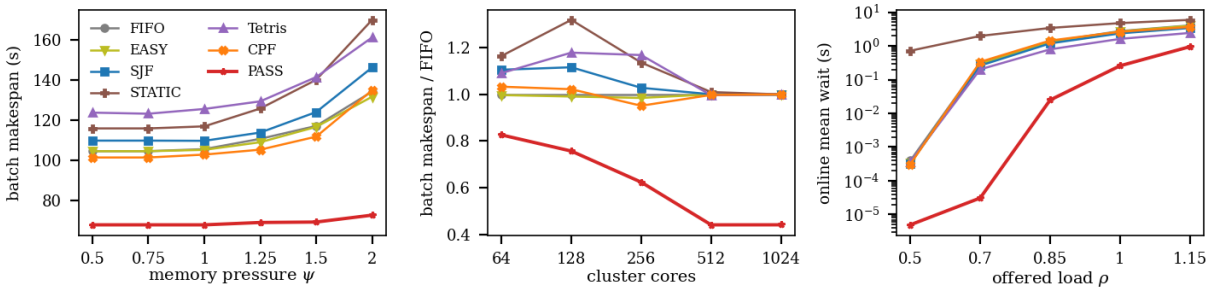


Figure 7. Platform and load sweeps (5 seeds): (a) batch makespan versus memory pressure ψ; (b) batch makespan normalized to FIFO versus cluster size for a fixed batch of 20 campaigns; (c) online mean waiting time versus offered load (log scale).

Memory pressure (Fig. 7a). With ample memory ($\psi \leq 0.75$) static requests never constrain packing: the baselines finish the batch in 101–124 s (STATIC: 116 s) and Pass in 67.9 s, a 33% reduction that stems entirely from speculation, while its mean waiting time is 0.77 s versus 0.72 s for the best baseline. As memory becomes scarcer, the baselines lose usable cores and their makespan grows to 132–170 s at $\psi = 2$, whereas Pass stays at 68–73 s because right-sized reservations keep every core usable. At $\psi = 2$ the reduction relative to the best baseline reaches 45% in makespan and 68% in mean waiting time.

Cluster size (Fig. 7b). For the same batch of 20 campaigns, the baselines benefit from additional cores only until they reach the longest chain: from 512 cores on, every baseline except STATIC finishes at exactly 100 s, the duration of the longest needed chain, and STATIC reaches this bound at 1024 cores. Pass breaks this barrier and finishes in 44 s on both 512 and 1024 cores (-56%), in line with the $(D + 1)$-fold chain acceleration of Proposition 2. On 64 cores, where the cluster is load-bound, Pass still reduces the makespan by 17% through right-sized packing and ordering.

Offered load (Fig. 7c). Under continuous arrivals all baselines start queueing at $\rho = 0.7$ (mean waits of 0.20–0.34 s, STATIC 1.97 s), whereas the mean waiting time of Pass is 0.00 s at $\rho = 0.7$ and 0.02 s at $\rho = 0.85$; at $\rho = 1.0$ and 1.15 it (0.26 and 0.95 s) remains 84% and 61% below the best baseline (Tetris). The campaign turnaround of Pass is 19–57% below the best baseline across the whole range. The benefit of speculation is largest at low load, where idle cores are abundant, and decreases as the cluster saturates; the waiting-time benefit of right-sizing grows with load.

7.5. Robustness

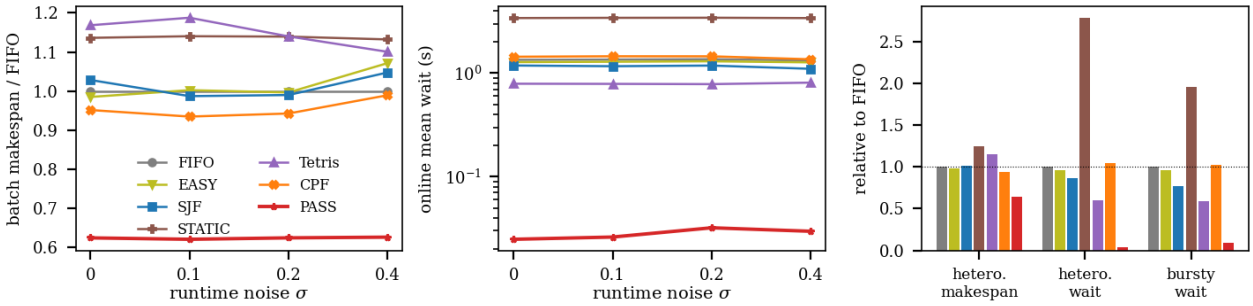


Figure 8. Robustness (5 seeds): (a) batch makespan and (b) online mean waiting time under additional lognormal runtime noise unknown to the schedulers; (c) node speed heterogeneity of $\pm 25\%$ and bursty arrivals (lognormal inter-arrival times with coefficient of variation 3), relative to FIFO.

Runtime noise. We multiply every runtime by an additional lognormal factor with σ up to 0.4, which the schedulers do not observe until a run completes. The batch makespan of Pass grows from 69.1 to 76.4 s and remains 37% below the best baseline at $\sigma = 0.4$ (CPF, 120.8 s), its batch mean waiting time stays at 0.84–0.86 s, and its online mean waiting time stays at 0.02–0.03 s, compared with 0.78–0.81 s for the best baseline (Fig. 8a,b).

Heterogeneous nodes and bursty arrivals. When node speeds vary uniformly by $\pm 25\%$, Pass keeps a 31% batch-makespan reduction (71.4 versus 104.0 s for CPF) and a 43% online-turnaround reduction (45.9 versus 80.0 s for Tetris). Under bursty arrivals, the mean waiting time of Pass changes from 0.09 to 0.22 s and remains 85% below the best baseline (Tetris, 1.48 s); its 95th-percentile waiting time is 0.78 s versus at least 5.7 s for all baselines, and the campaign turnaround is 31% lower (Fig. 8c).

7.6. Scheduling Overhead and Scalability

Table 7. Scheduling latency per decision (mean / 99th percentile, µs) for batch workloads scaled with the cluster (campaigns $=$ cores$/12.8$).

Cores	128	256	512	1024	2048
FIFO	9 / 42	13 / 37	19 / 49	35 / 85	64 / 154
SJF	29 / 86	68 / 169	196 / 881	534 / 1470	1672 / 3907
Tetris	68 / 148	95 / 220	134 / 306	250 / 535	489 / 1264
CPF	32 / 101	83 / 379	160 / 382	465 / 1172	1477 / 3826
Pass	126 / 496	178 / 736	242 / 822	415 / 1806	771 / 3718

Table 7 reports the per-decision latency of the schedulers as the cluster and the batch grow together from 128 to 2048 cores (up to 160 campaigns and 21,066 needed tasks). Pass needs 0.13–0.42 ms per decision up to 1024 cores and 0.77 ms (99th percentile 3.7 ms) at 2048 cores. Its cost grows sublinearly because only the chains of campaigns touched by an event are re-evaluated and speculation candidates are cached; from 1024 cores on, SJF and CPF become more expensive than Pass because they re-sort the whole ready queue with per-task estimates in every event. At 2048 cores the measured workload generates 312 scheduling events per second of simulated time, which Pass serves with a single core; for production analyses that run for minutes rather than seconds, the event rate and hence the relative overhead are proportionally lower.

8. Discussion and Conclusion

Deployment. Pass requires three inputs that structural simulation campaigns already produce: the planned number of integration steps of each run (record length and analysis step), a per-run collapse

flag (drift limit or non-convergence), and the measured runtime and peak memory of completed runs, which resource managers record through cgroup accounting. It can therefore be deployed either as a scheduling plugin of a resource manager or, without administrative privileges, as a user-level dispatcher that executes inside a pilot allocation [26] in the master–worker style used for parallel IDA [10]. Speculative runs need no special support: they are ordinary analyses whose results are kept or discarded by the dispatcher, and preemption is a process termination.

Generality. The scheduler only assumes that the successor of a task is needed with a probability that can be estimated online and that task cost is structured by a known size parameter. These assumptions also hold for hunt-and-fill IDA tracing, for adaptive multiple-stripe analysis that stops once the collapse fraction of a stripe saturates [22], for sequential uncertainty-propagation studies in which further samples are drawn only while a convergence criterion is unmet, and for pushover-then-response-history pipelines. Moldable execution of domain-decomposed OpenSees analyses can be integrated by treating the core count of a run as a further allocated resource whose speedup curve is estimated in the same campaign-scoped manner. Because Pass and PRESAGE [16] share the same probability-to-decision coupling, the two are complementary: the survival-weighted chain work of Pass can drive PRESAGE-style predictive elastic scaling, growing or shrinking a pilot allocation as the expected remaining work of the active campaigns changes.

Conclusion. We characterized 810 real OpenSees response-history analyses of four archetypes under the FEMA P695 far-field records and identified three properties of large-scale structural simulation campaigns—conditional chains whose parallelism decays as records collapse, runtimes structured by record length, model cost and intensity, and worst-case memory requests—that generic batch policies do not exploit. Following the uncertainty-to-decision principle of PRESAGE [16], Pass turns these properties into concrete scheduling mechanisms: an intensity-conditioned runtime model coupled with an online collapse-hazard estimator, an $O(1)$ survival-weighted chain-work metric, a bound-guided priority rule, probability-gated preemptible speculation of future intensity levels, and memory right-sizing with OOM feedback. In trace-driven experiments against seven baselines, Pass shortened the batch makespan by 33.9% and the online mean waiting time by 89.6% relative to the best baseline, raised core utilization from at most 57.8% to 74.6–79.5%, and remained effective under memory pressure, larger clusters, higher loads, runtime noise, heterogeneous nodes and bursty arrivals, at a decision latency of about 142µs. The released trace, simulator and scheduler provide a reproducible basis for further research on scheduling for computational earthquake engineering.

References

[1] Federal Emergency Management Agency. (2009). Quantification of building seismic performance factors (FEMA P695). Applied Technology Council.

[2] Vamvatsikos, D., & Cornell, C. A. (2002). Incremental dynamic analysis. Earthquake Engineering & Structural Dynamics, 31(3), 491–514.

[3] Vamvatsikos, D., & Cornell, C. A. (2004). Applied incremental dynamic analysis. Earthquake Spectra, 20(2), 523–553.

[4] McKenna, F., Scott, M. H., & Fenves, G. L. (2010). Nonlinear finite-element analysis software architecture using object composition. Journal of Computing in Civil Engineering, 24(1), 95–107.

[5] Zhu, M., McKenna, F., & Scott, M. H. (2018). OpenSeesPy: Python library for the OpenSees finite element framework. SoftwareX, 7, 6–11.

[6] McCallen, D., et al. (2021). EQSIM—a multidisciplinary framework for fault-to-structure earthquake simulations on exascale computers part I: Computational models and workflow. Earthquake Spectra, 37(2), 707–735. https://doi.org/10.1177/8755293020970982

[7] Zsarnóczay, A., et al. (2025). An open-source simulation platform to support and foster research collaboration in natural hazards engineering. Frontiers in Built Environment, 11, 1590479. https://doi.org/10.3389/fbuil.2025.1590479

[8] Lifka, D. A. (1995). The ANL/IBM SP scheduling system. In Job scheduling strategies for parallel processing (Vol. 949, pp. 295–303). Springer.

[9] Yoo, A. B., Jette, M. A., & Grondona, M. (2003). SLURM: Simple Linux utility for resource management. In Job scheduling strategies for parallel processing (Vol. 2862, pp. 44–60). Springer.

[10] Vamvatsikos, D. (2011). Performing incremental dynamic analysis in parallel. Computers & Structures, 89(1–2), 170–180.

[11] Tsafrir, D., Etsion, Y., & Feitelson, D. G. (2007). Backfilling using system-generated predictions rather than user runtime estimates. IEEE Transactions on Parallel and Distributed Systems, 18(6), 789–803.

[12] Li, J., Michelogiannakis, G., Cook, B., Cooray, D., & Chen, Y. (2023). Analyzing resource utilization in an HPC system: A case study of NERSC's Perlmutter. In High performance computing (pp. 242–257). Springer. https://doi.org/10.1007/978-3-031-32041-5_16

[13] Tovar, B., et al. (2018). A job sizing strategy for high-throughput scientific workflows. IEEE Transactions on Parallel and Distributed Systems, 29(2), 240–253.

[14] Bader, J., et al. (2024). Sizey: Memory-efficient execution of scientific workflow tasks. In 2024 IEEE International Conference on Cluster Computing (CLUSTER). https://doi.org/10.1109/CLUSTER59578.2024.00039

[15] Lehmann, F., et al. (2024). Ponder: Online prediction of task memory requirements for scientific workflows. In 20th IEEE International Conference on e-Science. https://doi.org/10.1109/e-Science62913.2024.10678682

[16] Wang, Y., Zou, X., Lu, Z., Wei, K., & Rhee, M. (2026). PRESAGE: Uncertainty-aware predictive elastic scheduling for cloud analytics and GenAI inference. IEEE Access, 14, 146019–146040. https://doi.org/10.1109/ACCESS.2026.3734498

[17] Grandl, R., Ananthanarayanan, G., Kandula, S., Rao, S., & Akella, A. (2014). Multi-resource packing for cluster schedulers. In Proceedings of the 2014 ACM SIGCOMM Conference (pp. 455–466).

[18] Topcuoglu, H., Hariri, S., & Wu, M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 13(3), 260–274.

[19] Ibarra, L. F., Medina, R. A., & Krawinkler, H. (2005). Hysteretic models that incorporate strength and stiffness deterioration. Earthquake Engineering & Structural Dynamics, 34(12), 1489–1511.

[20] Lignos, D. G., & Krawinkler, H. (2011). Deterioration modeling of steel components in support of collapse prediction of steel moment frames under earthquake loading. Journal of Structural Engineering, 137(11), 1291–1302.

[21] Haselton, C. B., Liel, A. B., Deierlein, G. G., Dean, B. S., & Chou, J. H. (2011). Seismic collapse safety of reinforced concrete buildings. I: Assessment of ductile moment frames. Journal of Structural Engineering, 137(4), 481–491.

[22] Baker, J. W. (2015). Efficient analytical fragility function fitting using dynamic structural analysis. Earthquake Spectra, 31(1), 579–599.

[23] Guan, X., & Burton, H. V. (2023). Collaborative filtering-based collapse fragility assessment. Earthquake Engineering & Structural Dynamics, 52(11), 3322–3344. https://doi.org/10.1002/eqe.3963

[24] Arroyo, O., Feliciano, D., Novoa, D., & Valcárcel, J. (2024). Opseestools: A Python library to streamline OpenSeesPy workflows. SoftwareX, 27, 101832.

[25] Boëzennec, R., et al. (2024). Qualitatively analyzing optimization objectives in the design of HPC resource manager. ACM Transactions on Modeling and Performance Evaluation of Computing Systems, 9, 1–28. https://doi.org/10.1145/3701986

[26] Merzky, A., Turilli, M., Titov, M., Al-Saadi, A., & Jha, S. (2022). Design and performance characterization of RADICAL-Pilot on leadership-class platforms. IEEE Transactions on Parallel and Distributed Systems, 33(4).

[27] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R., & Stoica, I. (2008). Improving MapReduce performance in heterogeneous environments. In Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (pp. 29–42).

[28] Zhang, Y., Zhang, T., Zhang, G., & Jacobsen, H.-A. (2023). Lifting the fog of uncertainties: Dynamic resource orchestration for the containerized cloud. In Proceedings of the 2023 ACM Symposium on Cloud Computing (SoCC) (pp. 48–64). https://doi.org/10.1145/3620678.3624646

[29] Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics, 17(2), 416–429.

[30] Kaplan, E. L., & Meier, P. (1958). Nonparametric estimation from incomplete observations. Journal of the American Statistical Association, 53(282), 457–481.