A. Discussion

Here we discuss some special cases and extensions of interest, including nonconvex models, infinite perturbations (e.g. $T=\infty$), and SGD.

A.1. Examples for Theorem 3.1

In this section, we discuss some examples where the bound (6) is applicable, along with some generalizations.

Example A.1 (Convex optimization)**.** Theorem 3.1 applies to ML systems that are based on minimizing a strongly convex objective. This includes many classical problems including regression.

Example A.2 (Nonconvex optimization)**.** If the loss function ℓ is nonconvex, then Theorem 3.1 still applies with some modifications. The assumptions (3) and (5) can be verified using known results on nonconvex optimization (Xu & Yin, 2017; Attouch et al., 2010) under the so-called *Kurdyka-Łojasiewicz property*, from which the bound (6) follows directly. Trouble arises, however, when ℓ has multiple basins of attraction: A perturbation δ_k could "push" the perturbed iterate $\widetilde{y}^{(k)}$ into a different basin, resulting in a limit point that is different from x^*. Theorem 3.1 continues to hold as long as this can be avoided, i.e. the δ_k are not too large.

Example A.3 (SGD)**.** The assumption (5) does *not* hold for SGD, which has a sublinear convergence rate in general. Nonetheless, it is straightforward to extend our framework to sublinear algorithms, with the caveat that analogous bounds on the iteration cost become more complicated. In fact, it is not hard to see from our proof how to do this: Lemma B.1 in the Appendix establishes the following useful general inequality

$$\mathbb{E}\|y^{(k+1)}-x^*\| \le c^{k+1}\left[\|x^{(0)}-x^*\|+\Delta_T\right].$$

Evidently, the factor of c governs how quickly Δ_T (i.e. the cost incurred by perturbations) gets washed out as k increases. For algorithms that converge sublinearly such as SGD, this effect will also be sublinear, but still tend to zero as long as the perturbations are not too large (see Appendix A.3 for a brief discussion). This is further corroborated by the empirical experiments in Section 5, where we show that the strategies for checkpoint-based fault tolerance proposed in the next section are successful on SGD as well as other optimization schemes such as alternating least squares. Similar arguments apply to convex (but not strongly convex) loss functions, for which gradient descent has a sublinear convergence rate in general.

Example A.4 (Infinite perturbations)**.** An interesting case occurs when $\delta_k \neq 0$ for all k. In other words, there is a possibility of a fault in *every iteration*. For arbitrary δ_k, it is clearly impossible to establish any kind of convergence result. In fact, suppose $\|\delta_k\| \le \Delta$ for each k. Then there is an irreducible error of $(c/(1-c))\Delta$, meaning that we cannot hope to obtain an ε-optimal solution for any $\varepsilon < (c/(1-c))\Delta$. This helps to explain why we focus on the nontrivial case with $\delta_k = 0$ for $k > T$ in Theorem 3.1. One setting in which the analysis with infinite perturbations is nontrivial is when Δ is known to be small, e.g. when using reduced precision as in Example 2.1. This setting can be analyzed by setting $\Delta \ge 2^{-(p-1)}\|x^{(k)}\|$ for all k. For details, see Appendix A.2.

A.2. Analysis for $T=\infty$

Suppose $\mathbb{E}\|\delta_k\| \le \Delta$ for each k. For intuition, note that Lemma B.1 implies that for any k,

$$\begin{aligned}\mathbb{E}\|y^{(k+1)}-x^*\| &\le c^{k+1}\left[\|x^{(0)}-x^*\|+\sum_{\ell=0}^{k} c^{-\ell}\Delta\right] \\ &= c^{k+1}\left[\|x^{(0)}-x^*\|+\Delta\frac{1-c^{-(k+1)}}{1-c^{-1}}\right] \qquad (7)\\ &= c^{k+1}\|x^{(0)}-x^*\|+\Delta\frac{c-c^{k+2}}{1-c} \\ &\xrightarrow{k\to\infty} \frac{c}{1-c}\Delta.\end{aligned}$$

Evidently, there is an irreducible, positive error if we are subjected to faults in every single iteration.

Thus, the best we can hope for is convergence to within some tolerance $\varepsilon > (c/(1-c))\Delta$. Re-arranging and solving for k in (7) as in the proof of Theorem 3.1, we deduce that $\mathbb{E}\|y^{(k+1)}-x^*\| < \varepsilon$ as long as

$$k > \frac{\log\left(\frac{\|x^{(0)}-x^*\|-\frac{c}{1-c}\Delta}{\varepsilon-\frac{c}{1-c}\Delta}\right)}{\log(1/c)}.$$

The resulting iteration cost bound is (cf. (6)):

$$\pi(\delta_k,\varepsilon) \le \frac{\log\left(\frac{1-\frac{\frac{c}{1-c}\Delta}{\|x^{(0)}-x^*\|}}{1-\frac{\frac{c}{1-c}\Delta}{\varepsilon}}\right)}{\log(1/c)}. \qquad (8)$$

This bound is only informative if $\|x^{(0)}-x^*\| > (c/(1-c))\Delta$ and $\varepsilon > (c/(1-c))\Delta$.

A.3. Stochastic gradient descent

Assume the objective function ℓ is strongly convex. In order to derive upper bounds on the iteration cost for SGD, we start from following general recursion, which is standard from the literature (Nemirovski et al., 2009; Rakhlin et al., 2012):

$$\mathbb{E}\|x^{(k+1)}-x^*\|^2 \le (1-\alpha_k)\mathbb{E}\|x^{(k)}-x^*\|^2+\alpha_k^2 G^2, \qquad (9)$$

where $\alpha_k \to 0$ is a sequence that depends on ℓ and the step size, and G is an upper bound on the expected norm of the stochastic gradients. Comparing (9) to (11), the only difference is that instead of a constant $c<1$, we have a sequence $1-\alpha_k \to 1$. Thus, instead of decaying at the geometric rate c^k, the iterates of SGD converge at a slower rate $(1-\alpha_1)\cdots(1-\alpha_k)$.

Define $a_k := (1-\alpha_1)\cdots(1-\alpha_k)$. Under the assumptions of

Theorem 3.1, we have the following analogue of (14):

$$\mathbb{E}\|y^{(k)}-x^*\| \le a_k\left[\|x^{(0)}-x^*\|+\sum_{\ell=0}^{T} a_\ell^{-1}\left(\mathbb{E}\|\delta_\ell\|+\alpha_\ell^2 G^2\right)\right] < \varepsilon.$$

This yields an implicit formula for k, which can be used to upper bound the iteration cost for SGD. For example, a popular choice of α_k is $\alpha_k \propto 1/k$, in which case $a_k \propto 1/k$ (this follows from an induction argument), and solving for k yields the desired upper bound.

B. Proofs

B.1. Proof of Theorem 3.1

We start with the following useful lemma:

Lemma B.1. *Assuming (5), we have for any k*

$$\mathbb{E}\|y^{(k+1)}-x^*\| \le c^{k+1}\left[\|x^{(0)}-x^*\|+\sum_{\ell=0}^{k} c^{-\ell}\mathbb{E}\|\delta_\ell\|\right]. \quad (10)$$

Proof. For any $k>0$ we have

$$\begin{aligned}\mathbb{E}\|y^{(k+1)}-x^*\| &= \mathbb{E}\|f(\widetilde{y}^{(k)})-x^*\| \\ &\le c\mathbb{E}\|\widetilde{y}^{(k)}-x^*\| \\ &= c\mathbb{E}\|y^{(k)}+\delta_k-x^*\| \\ &\le c\left[\mathbb{E}\|y^{(k)}-x^*\|+\mathbb{E}\|\delta_k\|\right], \quad (11)\end{aligned}$$

where we have invoked (5). Iterating this inequality, we obtain:

$$\begin{aligned}&c\left[\mathbb{E}\|y^{(k)}-x^*\|+\mathbb{E}\|\delta_k\|\right] \quad (12)\\ &\le c^2\mathbb{E}\|y^{(k-1)}-x^*\|+c^2\mathbb{E}\|\delta_{k-1}\|+c\mathbb{E}\|\delta_k\| \\ &\vdots \\ &\le c^{k+1}\mathbb{E}\|y^{(0)}-x^*\|+\sum_{i=0}^{k} c^{i+1}\mathbb{E}\|\delta_{k-i}\| \\ &= c^{k+1}\|x^{(0)}-x^*\|+\sum_{\ell=0}^{k} c^{k-\ell+1}\mathbb{E}\|\delta_\ell\|. \quad (13)\end{aligned}$$

In the last step we simply re-indexed the summation and use $y^{(0)}=x^{(0)}$. Combining (11) and (13) yields the desired bound. □

Proof of Theorem 3.1. By Lemma B.1, we have for any $k>T$,

$$\mathbb{E}\|y^{(k)}-x^*\| \le c^k\left[\|x^{(0)}-x^*\|+\sum_{\ell=0}^{T} c^{-\ell}\mathbb{E}\|\delta_\ell\|\right] < \varepsilon \quad (14)$$

$$\iff \frac{1}{\varepsilon}\left[\|x^{(0)}-x^*\|+\Delta_T\right] < c^{-k} \quad (15)$$

Re-arranging, we deduce that $\mathbb{E}\|y^{(k)}-x^*\|<\varepsilon$ if

$$k > \frac{\log\left(\frac{1}{\varepsilon}\left[\|x^{(0)}-x^*\|+\Delta_T\right]\right)}{\log(1/c)} \ge \kappa(y^{(k)},\varepsilon).$$

It is easy to check (e.g. take $\delta_k=0$ in the previous derivation) that $\kappa(x^{(k)},\varepsilon)=\log\left(\frac{1}{\varepsilon}\|x^{(0)}-x^*\|\right)/\log(1/c)$ is a bound on the number of iterations required for the unperturbed sequence $x^{(k)}$ to reach ε-optimality. Thus, the iteration cost is given by

$$\begin{aligned}\pi(\delta_k,\varepsilon) &= \kappa(y^{(k)},\varepsilon)-\kappa(x^{(k)},\varepsilon) \\ &\le \frac{\log\left(\frac{1}{\varepsilon}\left[\|x^{(0)}-x^*\|+\Delta_T\right]\right)-\log\left(\frac{1}{\varepsilon}\|x^{(0)}-x^*\|\right)}{\log(1/c)} \\ &= \frac{\log\left(1+\frac{\Delta_T}{\|x^{(0)}-x^*\|}\right)}{\log(1/c)},\end{aligned}$$

as claimed. □

B.2. Proof of Theorem 4.1

Let $z=x^{(C)}$ be the checkpoint of the model parameters saved at iteration C, and let S be the subset of model parameters lost during a failure at iteration T. Then

$$||\delta|| = ||z-x^{(T)}||$$

is the perturbation due to full recovery, and

$$||\delta'|| = ||z_S-x_S^{(T)}||$$

is the perturbation due to partial recovery, since $x_{S^c}^{(T)}$ does not change due to failure, where S^c is the complement set of S. Then we have

$$\begin{aligned}||\delta'||^2 &= ||z_S-x_S^{(T)}||^2 \\ &\le ||z_S-x_S^{(T)}||^2+||z_{S^c}-x_{S^c}^{(T)}||^2 \\ &\quad +(z_S-x_S^{(T)})\cdot(z_{S^c}-x_{S^c}^{(T)}) \\ &\le ||(z_S-x_S^{(T)})+(z_{S^c}-x_{S^c}^{(T)})||^2 \\ &= ||z-x^{(T)}||^2 = ||\delta||^2\end{aligned}$$

Thus $\|\delta'\|\le\|\delta\|$, as claimed.

B.3. Proof of Theorem 4.2

Let $z=x^{(C)}$ be the checkpoint of the model parameters saved at iteration C, and let S be the subset (chosen uniformly at random) of model parameters lost during a failure at iteration T. Then

$$\begin{aligned}\mathbb{E}||\delta'||^2 &= \mathbb{E}||z_S-x_S^{(T)}||^2 \\ &= \mathbb{E}\left[(z_S-x_S^{(T)})\cdot(z_S-x_S^{(T)})\right] \\ &= \sum_i \mathbb{E}\left[(z_S-x_S^{(T)})_i^2\right] \\ &= \sum_i \mathbb{E}\left[[i\in S](z_i-x_i^{(T)})^2\right] \\ &= \sum_i P(i\in S)(z_i-x_i^{(T)})^2 \\ &= \sum_i p(z_i-x_i^{(T)})^2 \\ &= p||z-x^{(T)}||^2 = p||\delta||^2\end{aligned}$$

Thus $\mathbb{E}||\delta'||^2=p||\delta||^2$, as claimed.

C. Implementation and experiments

C.1. Details of SCAR implementation

When a checkpoint is triggered:

1. The checkpoint coordinator sends a message to each PS node, which computes the distance of each of its parameters from their previously saved values in the running checkpoint using its in-memory cache.
2. Each PS node sends its model parameter IDs and computed distances to the checkpoint coordinator.
3. Upon receipt of the computed distances from all PS nodes, the checkpoint coordinator selects the fraction r of parameters with the largest distances, and sends their IDs back to their corresponding PS nodes.
4. Upon receipt of the parameter IDs, each PS node updates its in-memory cache, and saves those parameters to the shared persistent storage.

During step 4, the training algorithm can be resumed as soon as the in-memory caches have been updated, while output to the shared persistent storage happens asynchronously in the background. Thus, the checkpointing overhead in SCAR is just the time needed for prioritizing parameters and updating the in-memory cache. Furthermore, steps 2 and 3 simply answer a distributed top-k query. Although we chose a simple implementation for our prototype, a more scalable algorithm such as TPUT (Cao & Wang, 2004) can be used to remove the bottleneck of centralizing this computation onto the checkpoint controller.

When a failure is detected:

1. The failure detector notifies the recovery coordinator, which determines how the parameters belonging to the failed PS nodes should be re-partitioned.
2. The recovery coordinator partitions and sends the failed parameter IDs to the remaining PS ndoes, which re-load the parameters from the current running checkpoint in shared persistent storage.

SCAR is implemented using C++ and leverages an existing elastic ML framework (Qiao et al., 2018), which provides mechanisms for transparently re-routing requests from workers away from failed PS nodes, as well as for new PS nodes to join the active training job, replacing the old failed PS nodes.

C.2. Details of Models and Datasets in Section 5.1

Multinomial Logistic Regression (MLR). The model parameters are an $M \times N$ matrix of real numbers, where M is the dimensionality of the data, and N is the number of output classes. When distributed, the rows of the parameter matrix are randomly partitioned. For MNIST, we use a batch size of 10,000, a learning rate of 1×10^{-5}, and a convergence criteria of 2.5×10^4 in cross-entropy loss. For CoverType, we use a batch size of 1,000, a learning rate of 1×10^{-7}, and a convergence criteria of 6.7×10^5 in cross-entropy loss. For both datasets, the convergence criteria is reached in roughly 60 iterations.

Matrix Factorization (MF). The model parameters are matrices $L \in \mathbb{R}^{m \times p}$ and $R \in \mathbb{R}^{p \times n}$. When distributed, the rows of L and the columns of R are randomly partitioned. For MovieLens, we use 20 factors and a convergence criteria of 9.2×10^2 in mean squared error loss. For Jester, we use 5 factors and a convergence criteria of 5.57×10^3 in mean squared error loss. The MovieLens dataset is the `movielens-small` version consisting of 671 users and 9,125 items. The Jester dataset is the `Jester 2+` version, We further remove users with no ratings, and re-scale ratings from $[-10,10]$ to $[0,10]$. For both datasets, the factor matrices L and R are randomly initialized with each entry sampled uniformly at random from $[0,1)$, and the convergence criteria is reached in roughly 60 iterations.

Latent Dirichlet Allocation (LDA). The model parameters are the document-topic and word-topic distributions. We use a scaled total variation between document-topic distributions as the norm for computing distances between parameters. When distributed, the document-topic distributions are randomly partitioned across nodes. We do not consider failures of word-topic distributions because they can be re-generated from the latent token-topic assignments.

In LDA, each document in the input data consists of a series of tokens, where each token is assigned a categorical topic. Topic assignments are repeatedly randomly sampled during the lifetime of a job. From these token-topic assignments, a document-topic distribution is constructed for each document, and a word-topic distribution is constructed for each unique word. Both the document-topic and word-topic distributions can be re-generated given the token-topic assignments, so losing the distributions themselves is not a problem. However, when distributed, each document-topic distribution is typically co-located with the document it corresponds to. Thus, losing a document-topic distribution is typically associated with also losing the token-topic assignments of that document, which do require recovery from a saved checkpoint. Therefore, we only consider the loss of document-topic distributions, and assume that the word-topic distributions can be reconstructed at any time.

Since the parameters of LDA are distributions, a natural norm to use is the total variation norm. However, the total variation norm when applied to LDA puts the same weight onto every document-topic distribution. This means that re-sampling a token-topic assignment in a shorter document has a greater impact to the overall norm than re-sampling a token-topic assignment in a longer document, which biases checkpoint prioritization towards shorter documents. To address this, we scale the total variation norm of each document-topic distribution by the length of the document it corresponds to. The result is still a valid norm, since it is a positive linear combination (which is constant with respect to the input data) of total variation norms.

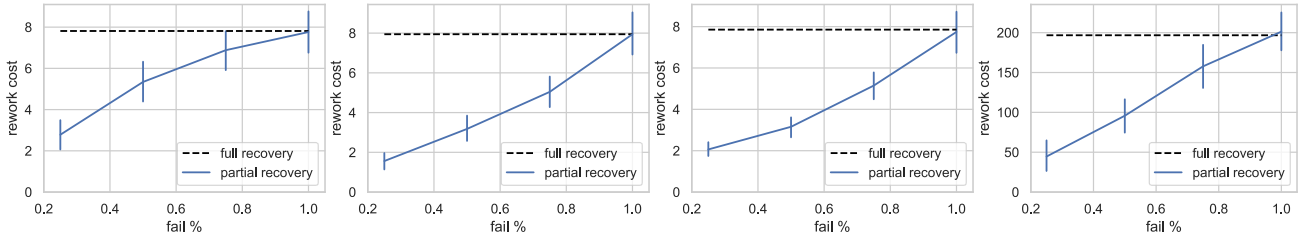
For 20 Newsgroups, we use a convergence criteria of 9.5×10^6 in negative log-likelihood. For Reuters, we use a convergence criteria of 8.5×10^5 in negative log-likelihood. For both datasets

we train using 20 topics and hyperparameters $\alpha = \beta = 1$. The convergence criteria is reached in roughly 60 iterations.

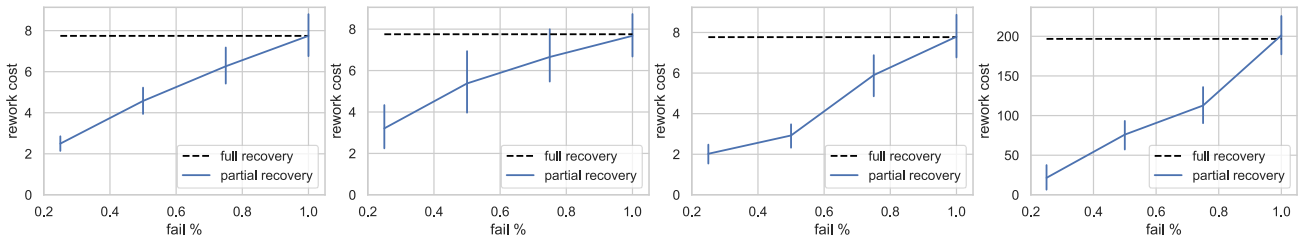
Convolutional Neural Network (CNN). The network consists of 2 convolution layers with ReLU activations (Nair & Hinton, 2010) and max pooling followed by 3 fully-connected layers with ReLU activation. Because of the structure in neural network models, we consider two different partitioning strategies: 1) In *by-layer* partitioning, we assume that the layers of the network are randomly partitioned across nodes; and 2) In *by-shard* partitioning, we further divide each layer's parameters into shards, and all shards are randomly partitioned across nodes. The experiments shown in Sec. 5 use by-layer partitioning, but we show both in Fig. C.1 and Fig. C.2. We use a batch size of 64, the recommended Adam settings of $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$, and a convergence criteria of 0.08 in cross-entropy loss. In by-layer partitioning, the weight and bias parameters are independent and partitioned separately (so they can either be lost together, or not). In by-shard partitioning, each parameter tensor is evenly partitioning according to its first dimension. The optimizer parameters (ie. the running first and second moment estimates in the case of Adam) are placed with their corresponding model parameters. Thus, an optimizer parameter is always faulted simultaneously with its corresponding model parameter.

C.3. Additional experiments

Figures C.1 and C.2 show additional results on the same four models applied to different datasets. The first row in each figure is the same as Figures 6 and 7 in the main paper.

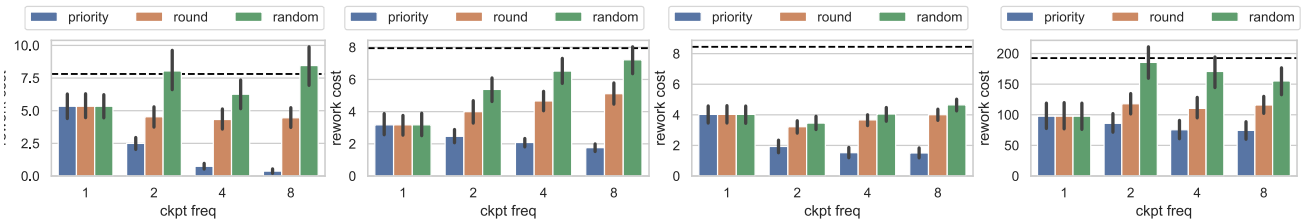


(a) MLR on CoverType. (b) MF on MovieLens. (c) LDA on 20 Newsgroups. (d) CNN-bylayer on MNIST.

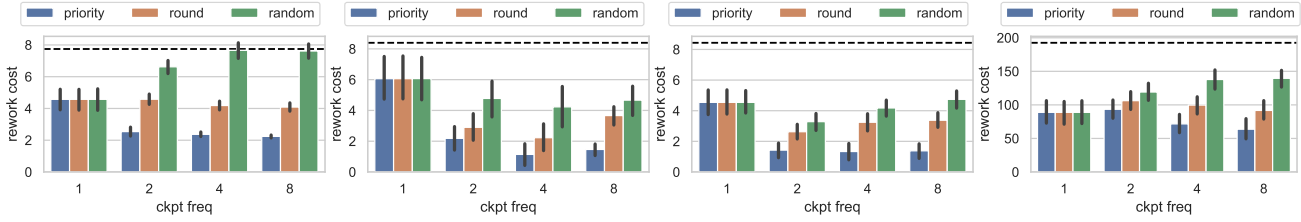


(e) MLR on MNIST. (f) MF on Jester. (g) LDA on Reuters. (h) CNN-byshard on MNIST.

Figure C.1. Partial vs. full recovery for all models and datasets, where the set of failed parameters are selected uniformly at random. The x-axis shows the fraction of failed parameters, and the y-axis shows the number of rework iterations. The error bars indicate 95% confidence intervals, calculated by repeating each trial 100 times.



(a) MLR on CoverType. (b) MF on MovieLens. (c) LDA on 20 Newsgroups. (d) CNN-bylayer on MNIST.



(e) MLR on MNIST. (f) MF on Jester. (g) LDA on Reuters. (h) CNN-byshard on MNIST.

Figure C.2. Prioritized checkpoint experiments for all models and datasets comparing between the random, round-robin, and priority strategies. The x-axis indicated checkpoint frequency relative to full checkpoints, where 1 indicates full checkpoints, 2 indicates $1/2$ checkpoints at $2\times$ frequency, etc., and the y-axis shows the number of rework iterations. The error bars indicate 95% confidence intervals, calculated by repeating each trial 100 times, and the dashed black line represents the rework cost of a full checkpoint.